E. E. Holmes and E. J. Ward

Analysis of multivariate time-series using the MARSS package

April 20, 2010

Mathematical Biology Program
Northwest Fisheries Science Center

Holmes, E. E. and E. J. Ward. 2009. Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112, unpublished documentation. Contact eli.holmes@noaa.gov

Preface

The motivation for our work with MARSS models was a collaboration with Rich Hinrichsen (Hinrichsen and Holmes, 2009). Rich developed a framework for analysis of multi-site population count data using MARSS models and bootstrap AICb. Our work (EEH and EJW) extended Rich's framework, made it more general, and led to the development of a parametric bootstrap AICb for MARSS models, which allows one to do model-selection using datasets with missing values(Ward et al., 2009; **?**). Later, we developed additional algorithms for simulation and confidence intervals. Discussions with Mark Scheuerell led to an extensive revision of the EM algorithm and to the development of a general EM algorithm for constrained MARSS models(Holmes, 2010). Discussions with Mark also led to a complete rewrite of the model specification so that the package could be used for MARSS in general – rather than simply the form of MARSS model used in our (EEH and EJW) applications. Many collaborators have helped testing the package; we thank especially Yasmin Lucero, Mark Scheuerell, Kevin See, and Brice Semmens. Development of the code into a R package would not have been possible without Kellie Wills, who developed the package and wrote the majority of the code outside of the algorithm functions.

The case studies used in this manual were developed for workshops on analysis of multivariate time series data given at the Ecological Society meetings since 2005 and taught by us (EEH and EJW) along with Yasmin Lucero, Stephanie Hampton, Brice Semmens, and Mark Scheuerell. The case study on extinction estimation and trend estimation was initially developed by Brice Semmens and later extended by us for this manual. The algorithm behind the TMU figure in case study 1 was developed during a collaboration with Steve Ellner (Ellner and Holmes, 2008).

EEH and EJW are research scientists at the Northwest Fisheries Science Center in the Mathematical Biology program. This work was conducted as part of our jobs at the Northwest Fisheries Science Center, a research center for NOAA Fisheries which is a US federal government agency. A CAMEO grant from NOAA Fisheries supported Kellie Wills. During the initial stages of this work, EJW was supported on a post-doctoral fellowship from the National Research Council.

You are welcome to use the code and adapt it with attribution. It may not be used in any commercial applications. Links to more code and publications on MARSS applications can be found by following the links at EEH's website http://faculty.washington.edu/eeholmes Links to our papers that use these methods can also be found at the same website.

Contents

1

The MARSS package

The MARSS package is designed to fit constrained and unconstrained linear MARSS models of the form

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t, \text{ where } \mathbf{W}_t \sim \text{MVN}(0, \mathbf{Q}) \tag{1.1a}$$

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t, \text{ where } \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R}) \tag{1.1b}$$

$$\mathbf{x}_1 \sim \text{MVN}(\pi, \mathbf{V}_1) \tag{1.1c}$$

The model includes random variables, parameters and data:

- $\mathbf{x}_t$ is a $m \times 1$ column vector of the hidden states at time t. It is a realization of the random variable $\mathbf{X}_t$.
- $\mathbf{w}_t$ is a $m \times 1$ column vector of the process errors at time t. It is a realization of the random variable $\mathbf{W}_t$.
- $\mathbf{y}_t$ is a $n \times 1$ column vector of the observed data at time t. Missing values are allowed.
- $\mathbf{v}_t$ is a $n \times 1$ column vector of the non-process errors at time t. It is a realization of the random variable $\mathbf{V}_t$.
- $\mathbf{B}$ is a parameter and is a $m \times m$ matrix.
- $\mathbf{u}$ is a parameter and is a $m \times 1$ column vector.
- $\mathbf{Q}$ is a parameter and is a $m \times m$ variance-covariance matrix.
- $\mathbf{Z}$ is a parameter and is a $n \times m$ matrix.
- $\mathbf{a}$ is a parameter and is a $n \times 1$ column vector.
- $\mathbf{R}$ is a parameter and is a $n \times n$ variance-covariance matrix.

The meaning of the parameters depends on the application for which the MARSS model is being used. In the case studies, we show examples of MARSS used as population models. However, MARSS models are widely used in many fields, engineering, finance, genetics, physics, etc., and our examples are just one way that MARSS models are used. The MARSS package is not specific to population modeling applications. It fits generic MARSS models of the form in Equation (1.1).

The MARSS package allows one to fit both unconstrained models and models in which the parameters are constrained in the sense that they have fixed, free and shared values. For example, let $\mathbf{M}$ and $\mathbf{M}$ be arbitrary matrix and column vector parameters. The MARSS package allows one to easily specify and fit models where $\mathbf{M}$ and $\mathbf{M}$ have the following forms.

$$\mathbf{M} = \begin{bmatrix} a & 0.9 & c \\ -1.2 & a & 0 \\ 0 & c & b \end{bmatrix} \text{ and } \mathbf{M} = \begin{bmatrix} d \\ d \\ e \\ 2.2 \end{bmatrix}$$

Version 1.0 of the MARSS package fits models via maximum-likelihood[1] using an EM algorithm. It also supplies functions for various standard frequentist computations: confidence intervals, bootstrapping, model selection (via AIC and parametric and non-parametric bootstrap AIC), simulation, and bootstrap bias correction. Version 1.0 does not allow $\mathbf{B}$ or $\mathbf{Z}$ to be estimated and $\mathbf{a}$ is constrained to act as a scaling factor. Version 2.0 is currently being tested and it will allow $\mathbf{B}$, $\mathbf{Z}$, and $\mathbf{a}$ estimation along with less constrained forms of $\mathbf{Q}$ and $\mathbf{R}$. Version 3.0 is in development and will provide Bayesian estimation. The EM algorithm is used because it is robust to missing values and to models with various constraints, however quasi-Newton methods might also work in cases where there are no missing values. The `DLM` package (search for it on CRAN) provides fitting via quasi-Newton methods; the MARSS package provides a function for converting MARSS model objects to DLM model objects.

[1] Be aware ML estimates of variance in MARSS models are fundamentally biased. This bias is more severe when one or the other of $\mathbf{R}$ or $\mathbf{Q}$ is very small and the bias does not go to zero as sample goes to infinity. You can generate an unbiased using a bootstrap estimate bias. The function MARSSparamCIs() will do this. However be aware that adding an *estimated* bias to a parameter estimate will lead to an increase in the variance of your parameter estimate. The amount of variance added will depend on sample size.

2

The main MARSS functions

The MARSS 1.0 functions use maximum-likelihood to fit models to data. They work with and produce two main types of objects: a marss model which is class=marssm and a maximum-likelihood fitted model object which is class=marssMLE. A marss model object specifies the structure of the model to be fitted. It is an R code version of the model. Below "modleObj" means the argument (or returned value is a marss model object, and "MLEobj" means the argument (or returned value) is a marssMLE object.

2.1 Fitting a MARSS model to data

`MLEobj=PopMARSS(data)` This will fit a MARSS model to the data using a default model where the number of state processes is equal to the number of observation time series. The default has a diagonal observation error matrix and unconstrainted process error matrix. `MLEobj` is a marss object where the estimated parameters are in `MLEobj$params`. Type `summary(MLEobj)` to see the model and summary of the fitted parameters. The PopMARSS function takes care of error-checking and model structure. The rest of the fitting functions are core functions.

`MLEobj=MARSSkf(data, modelObj)` This will compute the expected values of the hidden states given data and a MARSS model object via the Kalman filter (to produce estimates conditioned on $1 : t-1$) and the Kalman smoother (to produce estimates conditioned on $1 : T$. The function also returns the exact likelihood of the data conditioned on the model using the innovations algorithm with missing value corrections (see Chapter 5).

`MLEobj=MARSSkem(MLEobj)` This will fit a MARSS model via the Kalman-EM algorithm to the data using a properly specified marss MLE object; this has data, the marss model and the necessary initial condition and control elements. See the appendix on the object structures in the MARSS package. `MARSSkem` does no error-checking. See `is.marssMLE()`.

`MLEobj=MARSSmcinit(MLEobj)` This will perform a Monte Carlo initial conditions search and update the marss MLE object with the best initial conditions from the search.

`is.marssMLE(MLEobj)` This will check that a marss mle object is properly specified and ready for fitting. This should be called before `MARSSkem` is called. This function is not typically needed if using `PopMARSS` since `PopMARSS` builds the marss model object for the user and does error-checking on model structure.

`is.marssm(modelObj)` This will check that the free and fixed matrices in a marss model object are properly specified. This function is not typically needed if using `PopMARSS` since `PopMARSS` builds the marss model object for the user and does error-checking on model structure.

2.2 Using a fitted marss object (`class=marssMLE`)

The following functions use objects of `class=marssMLE`. Type ?function.name to see information on usage and examples.

`MLEobj=MARSSaic(MLEobj)` This adds model selection criteria, AIC, AICc, and AICb, to a marssMLE object.

`boot=MARSSboot(MLEobj)` This returns bootstrapped parameters and data via parametric or innovations bootstrapping.

`MLEobj=MARSShessian(MLEobj)` This adds a numerically estimated Hessian matrix to a marssMLE object.

`MLEobj=MARSSparamCIs(MLEobj)` This adds standard errors, confidence intervals, and bootstrap estimated bias for the ML parameters using bootstrapping or the Hessian to the passed in marssMLE object.

`sims=MARSSsimulate(parList)` This returns simulated data from a MARSS model specified via the parameter matrices in `parList` (this is a list with elements `Q`, `R`, `U`, etc). Typically one would pass in `MLEobj$params` as one's parameter list, but you can also construct the list manually.

`paramVec=MARSSvectorizeparams(MLEobj)` This returns the estimated (and only the estimated) parameters as a vector. This is useful for storing the results of simulations and for writing functions that fit MARSS models using R's `optim` function. The function can also be used to assign free values to a MLE object. See ?MARSSvectorizeparams for other uses.

`newMLEobj=MARSSvectorizeparams(MLEobj, paramVec)` This returns a marssMLE object in which the estimated parameters (which are in `MLEobj$params` along with the fixed values) are replaced with the values in `paramVec`. This is useful when you have bootstrapped parameter sets. You can then create proper marssMLE objects from the bootstrapped parameters using `bootMLEobj=MARSSvectorizeparams(MLEobj, bootparamVec)`and simulate from those using `MARSSsimulate`.

3

MARSS model specification in the core functions

Most users will not directly work with the core functions nor build marss model objects from scratch. Instead, they will usually interact with the core function via a wrapper function such as PopMARSS described in chapter 4. However, a basic understanding of the structure of a marss model object is necessary if one wants to fit more flexible models or to interact with the core functions.

A MARSS model is fit to data by passing in data and a marss model object (class=marssm) to one of the MARSS fitting functions. A marss model object specifies the structure of the MARSS model (Equation 1.1) in R code. In this chapter, we describe how the model is specified for the core functions, but in most cases, it is better to access the core functions by using a wrapper function. In many applications, the MARSS model will take a very specific form, and the user only needs to change specific parameters or change specific structures (like “unconstrained” versus ”diagonal”). A wrapper function builds the basic model structure for the user and only changes the necessary elements or structures. The core functions are designed to be very flexible so that one can fit many different types of models, but that flexibility can be overwhelming if the user only needs to fit a specific model structure. Chapter 4 describes the wrapper function `PopMARSS` which is designed for analysis of population data using a multivariate Gompertz model.

The first step of model specification is to write down the model in matrix form (Equation 1.1) with notes on the dimensions (rows and columns) of each parameter and for $\mathbf{x}$ and $\mathbf{y}$. In the core functions, the parameters in the MARSS model must be passed as matrices of the correct dimension. Thus the parameters in the R functions correspond one-to-one to the mathematical equation. For example, $\mathbf{U}$ must be passed in as a matrix of dimension `c(m,1)`. The function will return an error if anything else is passed in (including a matrix with `dim=c(1,m)`).

3.1 Specifying the fixed and free components of the parameters

In a marss model object, each parameter must be specified by a pair of matrices: `free` which gives the location and sharing of the estimated elements in the parameter matrix and `fixed` which specifies the location and value of the fixed element in the parameter matrix. For example, **Q** is specified by `free$Q` and `fixed$Q`.

The fixed matrix specifies the values (numeric) of the fixed (meaning not estimated) elements. In the fixed matrix, the free (meaning estimated or fitted) elements are denoted with `NA`. The following shows some common examples of the fixed matrix using `fixed$Q` as the example. Each of the other fixed matrices for the other parameters uses the same pattern.

- **Q** is unconstrained, so there are no fixed values

$$\texttt{Q.fixed} = \begin{bmatrix} NA & NA & NA \\ NA & NA & NA \\ NA & NA & NA \end{bmatrix}$$

- **Q** is a diagonal matrix, so the off-diagonals are fixed at 0. The diagonal elements will be estimated.

$$\texttt{Q.fixed} = \begin{bmatrix} NA & 0 & 0 \\ 0 & NA & 0 \\ 0 & 0 & NA \end{bmatrix}$$

- **Q** is fixed, i.e. will not be estimated rather all values in the **Q** matrix are fixed.

$$\texttt{Q.fixed} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

The free matrix specifies which elements are estimated and specifies how (and whether) the free elements are shared. In the free matrix, the fixed elements are denoted `NA`. The following shows some common examples of `free` using `free$Q` as the example. Note that `free` can be either a character matrix or a numeric matrix.

- **Q** is a diagonal matrix in which there is only one shared value on the diagonal. Thus there is only one **Q** parameter.

$$\texttt{Q.free} = \begin{bmatrix} 1 & NA & NA \\ NA & 1 & NA \\ NA & NA & 1 \end{bmatrix} \quad \texttt{or} \quad \begin{bmatrix} "a" & NA & NA \\ NA & "a" & NA \\ NA & NA & "a" \end{bmatrix}$$

- **Q** is a diagonal matrix in which each of the diagonal elements are different.

$$\texttt{Q.free} = \begin{bmatrix} 1 & NA & NA \\ NA & 2 & NA \\ NA & NA & 3 \end{bmatrix} \quad \texttt{or} \quad \begin{bmatrix} "north" & NA & NA \\ NA & "middle" & NA \\ NA & NA & "south" \end{bmatrix}$$

- **Q** has one value on the diagonal and another one on the off-diagonals. There are no fixed values in **Q**.

$$\texttt{Q.free} = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \quad \texttt{or} \quad \begin{bmatrix} "a" & "b" & "b" \\ "b" & "a" & "b" \\ "b" & "b" & "a" \end{bmatrix}$$

- **Q** is unconstrained. There are no fixed values in **Q** in this case. Note that since, **Q** is a variance-covariance matrix, it must be symmetric across the diagonal.

$$\texttt{Q.free} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

  B and **Z** do not have that limitation, thus their unconstrained versions have all elements different:

$$\texttt{B.free} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

3.2 Limits on the forms of the parameter constraints (version 1.0)

MARSS 1.0 will allow any combination of fixed and shared values in **A** and **U**, but in **R**, **Q**, **B**, and **Z** there are limits to what forms these matrices can take. These limitations have to do with the way the EM algorithm is coded for version 1.0. Version 2.0 will remove many of these restrictions.

- **R** and **Q** can be fixed, unconstrained, diagonal with any pattern of shared values on the diagonal, a matrix with one value on the diagonal and another on the off-diagonals (an "equal var-cov" matrix), and block-diagonal in these patterns.
- If there are missing values in the data, **R** must be diagonal (or fixed).
- **B** can be fixed, unconstrained, diagonal, or block diagonal unconstrained. If **B** is diagonal, there can be any pattern of shared values or fixed values on the diagonal. However, if **B** is diagonal, **Q** must also be diagonal. This last constraint is one-way; if **Q** is diagonal, it is not necessary that **B** be diagonal.
- **Z** can be fixed, unconstrained, diagonal, or block diagonal unconstrained. If **Z** is diagonal, there can be any pattern of shared values or fixed values on the diagonal. However, if **Z** is diagonal, **R** must also be diagonal. This last constraint is one-way; if **R** is diagonal, it is not necessary that **Z** be diagonal.

The other limitation is that one must specify a model that has only one solution. The core MARSS functions will allow you to attempt to fit an improper model (one with multiple solutions). If you do this accidentally, it may or may not be obvious that you have a problem. The MARSS estimation functions may chug along happily and return a solution. Careful thought about your model structure and the structure of the estimated parameter matrices will help you determine if your model is under-constrained and unsolvable. Basically, take care when using MARSS core functions directly and remember that it will not prevent you from fitting an under-constrained model. This is not a problem when using PopMARSS – as long as you do not use its optional `fixed` or `free` arguments. The PopMARSS wrapper builds the marss model for the user and is written in such a way that it prevents users from specifying an under-constrained model.

4

The PopMARSS wrapper function

Wrapper functions are written to ease the construction of the `free` and `fixed` matrices when one is working with a MARSS model that takes specific forms. There is no particular way that the wrapper needs to be written; it is simply an R function that builds a marss model object for the user. PopMARSS is one example of a wrapper function. It is probably more complex than a wrapper function need be. It includes print functions and extensive checks for model consistency.

4.1 PopMARSS, a wrapper for multivariate Gompertz models

The PopMARSS function is designed for fitting multivariate Gompertz models:

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{U} + \mathbf{E}_{t-1}, \qquad \text{where } \mathbf{E}_{t-1} \sim MVN(0, \mathbf{Q}) \tag{4.1a}$$
$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{A} + \boldsymbol{\eta}_t, \qquad \text{where } \boldsymbol{\eta}_t \sim MVN(0, \mathbf{R}) \tag{4.1b}$$

The $\mathbf{y}$ is a $n \times T$ matrix and is interpreted as n time series from $t = 1 : T$ of logged population counts and the $\mathbf{x}$ are the m hidden population trajectories. For example, a $\mathbf{y}$ data matrix of 3 sites measured for 10 time steps would look like

$$\mathbf{y} = \begin{bmatrix} 1 & 2 & -99 & ... & 8 \\ 2 & 5 & 3 & ... & 5 \\ 1 & -99 & 2 & ... & 7 \end{bmatrix}$$

where -99 denotes a missing value. $\mathbf{x}$ might look like (here $m = 2$):

$$\mathbf{x} = \begin{bmatrix} 0.8 & 2.2 & 3 & ... & 7.1 \\ 1.5 & 2.5 & 2.5 & ... & 6 \end{bmatrix}$$

$\mathbf{Z}$ is a $n \times m$ design matrix of zeros and ones where the row sums equal 1. $\mathbf{Z}$ is specifying which observation time series, $y_{i,1:T}$, is associated with which population trajectory, $x_{j,1:T}$. $\mathbf{Z}$ is like a look up table with 1 row for each of the n observation time series and 1 column for each of the m population trajectories. A 1 in row i column j means that $\mathbf{y}$ time series i is measuring the j-th $\mathbf{x}$ trajectory. Otherwise the value in $\mathbf{Z}_{ij} = 0$. The case studies give many examples of different ways the multivariate Gompertz can be used to analyze population data. Here we only show how models are specified using `PopMARSS`.

In the `PopMARSS` function, the user specifies the model by passing in a parameter constraint list:

```
PopMARSS(data, constraint=list(Z=Z.constraint, B=B.constraint,
       U=U.constraint, Q=Q.constraint, A=A.constraint,
       R=R.constraint, x0=x0.constraint) )
```

`data` must be a $n \times T$ matrix, that is time goes across columns (ref. Equation 1.1). Note that there is no option to change the how the variance of the initial states is treated because this variance is set to 0 so that initial states are treated as fixed but unknown; the initial states are one of the estimated parameters.

The `constraint` options for each parameter are listed below. In all case, however, `*.constraint="ignore"` can be set. This means that the parameter `*` is specified by the user by setting `fixed=list(*=*.fixed)` and `free=list(*=*.free)`. Note that `*` is used here as the placeholder for the parameter name `Q, R, x0,` etc. It is necessary to set the constraint to `"ignore"` so that the default constraints are not applied.

4.2 Process equation constraints

4.2.1 `B.constraint`

$\mathbf{B}$ is a $m \times m$ matrix. In PopMARSS, $\mathbf{B}$ can take only certain constrained forms. Below we show the forms using $m = 3$ in our examples.

- `B.constraint="identity"` The $\mathbf{B}$ matrix is the identity matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- `B.constraint="diagonal and equal"` The $\mathbf{B}$ matrix is diagonal and has only 1 parameter:

$$\begin{bmatrix} b & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & b \end{bmatrix}$$

- `B.constraint="diagonal and unequal"` The **B** matrix is diagonal and has m parameters:
$$\begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix}$$
- `B.constraint="unconstrained"` There are m^2 **B** parameters:
$$\begin{bmatrix} b_1 & b_{1,2} & b_{1,3} \\ b_{2,1} & b_2 & b_{2,3} \\ b_{3,1} & b_{3,2} & b_3 \end{bmatrix}$$
- `B.constraint=as.factor(c(...))` When **B** is specified as a length m character or numeric vector of class factor (e.g. `as.factor(c(...))`, it indicates that **B** is diagonal and the vector of factors specifies which values on the diagonal are shared. For example, `B.constraint=as.factor(c(1,1,2))` means **B** takes the form
$$\begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_1 & 0 \\ 0 & 0 & b_2 \end{bmatrix}$$
There are two **B** parameters in this case. The factor levels can be either numeric or character. `c(1,1,2)` is the same as `c("north","north","south")`
- `B.constraint=matrix(..., nrow=m, ncol=m))` Passing in a $m \times m$ matrix, means that **B** is fixed to the values in the matrix. The matrix must be numeric.

4.2.2 U constraints

The **U** constraint has the following options:

- `U.constraint="equal"` There is only **U** parameter.
$$\begin{bmatrix} u \\ u \\ u \end{bmatrix}$$
- `U.constraint="unequal"` or `U.constraint="unconstrained"` These are equivalent. There are m **U** parameters.
$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$
- `U.constraint=as.factor(c(...))` The **U** constraint is specified as a length m character or numeric vector of class factor. The vector of factors specifies which values in **U** are shared. For example, `U.constraint=as.factor(c(1,1,2))` means that **U** has the following structure:

$$\begin{bmatrix} u_1 \\ u_1 \\ u_2 \end{bmatrix}$$

There are two **U** parameters in this case. The factor levels can be either numeric or character. `c(1,1,2)` is the same as `c("north","north","south")`.

- `U.constraint=matrix()` Passing in a $m \times 1$ matrix, means that **U** is fixed to the values in the matrix. The matrix must be numeric. In MARSS version 1.0, **U** cannot varying in time, even if fixed.

4.2.3 Q constraint

The **Q** constraint has the following options:

- `Q.constraint="diagonal and equal"` There is only one process variance term in this case.
$$\begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}$$
- `Q.constraint="diagonal and unequal"` There are m process variance parameters in this case.
$$\begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{bmatrix}$$
- `Q.constraint="unconstrained"` There are values on the diagonal and off-diagonals of **Q** and variance and covariance is different.
$$\begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \sigma_{1,3} \\ \sigma_{1,2} & \sigma_2^2 & \sigma_{2,3} \\ \sigma_{1,3} & \sigma_{2,3} & \sigma_3^2 \end{bmatrix}$$
There are m process variance parameters and $(m^2 - m)/2$ covariances in this case, so $(m^2 + m)/2$ parameters.
- `Q.constraint="equalvarcov"` There is one process variance parameter and one covariance, so 2 parameters.
$$\begin{bmatrix} \sigma^2 & \sigma_{i,j} & \sigma_{i,j} \\ \sigma_{i,j} & \sigma^2 & \sigma_{i,j} \\ \sigma_{i,j} & \sigma_{i,j} & \sigma^2 \end{bmatrix}$$
- `Q.constraint=as.factor(c(...))` The **Q** constraint is specified is a length m character or numeric vector of class factor. This means that **Q** is diagonal and the vector of factors specifies which values on the diagonal are shared. For example, `Q.constraint=as.factor(c(2,1,2))` means that **Q** takes the form:

$$\begin{bmatrix} \sigma_2^2 & 0 & 0 \\ 0 & \sigma_1^2 & 0 \\ 0 & 0 & \sigma_2^2 \end{bmatrix}$$

`Q.constraint=as.factor(c(1,1,2))` means that **Q** takes the form:

$$\begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_1^2 & 0 \\ 0 & 0 & \sigma_2^2 \end{bmatrix}$$

The factor levels can be either numeric or character. `c(1,1,2)` is the same as `c("north","north","south")`.

- `Q.constraint=matrix(..., nrow=m, ncol=m))` Passing in a $m \times m$ matrix, means that **Q** is fixed to the values in the matrix. The matrix must be numeric.

4.2.4 x initial condition constraints

The constraints on the initial conditions of **x** has the following options:

- `x0.constraint="equal"` There is only initial state parameter.

$$\begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix}$$

- `x0.constraint="unequal"` or `x0.constraint="unconstrained"` These are equivalent. There are m initial state parameters.

$$\begin{bmatrix} x_{1,0} \\ x_{2,0} \\ x_{3,0} \end{bmatrix}$$

- `x0.constraint=as.factor(c(...))` The initial states constraint is specified is a length m character or numeric vector of class factor. The vector of factors specifies which initial states have the same value. For example, `x0.constraint=as.factor(c(1,1,2))` means that the initial states have the following structure:

$$\begin{bmatrix} x_{1,0} \\ x_{1,0} \\ x_{2,0} \end{bmatrix}$$

There are two initial state parameters in this case. The factor levels can be either numeric or character. `c(1,1,2)` is the same as `c("north","north","south")`.

- `x0.constraint=matrix(..., nrow=m, ncol=1))` Passing in a $m \times 1$ matrix, means that the initial states are fixed to the values in the matrix. The matrix must be numeric.

4.3 Observation equation constraints

4.3.1 Z constraint

In PopMARSS, $\mathbf{Z}$ is a $n \times m$ matrix that specifies which x_i hidden state time series correspond to which y_j time series. Each y_j time series (each row in $\mathbf{y}$) corresponds to one and only one x_i time series (row in $\mathbf{x}$). In PopMARSS, the $\mathbf{Z}$ constraint is normally specified as a length n vector of class factor. The i-th element of this vector specifies which population trajectory the i-th observation time series belongs to. Here are some examples; see the case studies for more examples.

- `Z.constraint=as.factor(c(1,1,1))` All $\mathbf{y}$ time series are observing the same (and only) hidden state trajectory x. Thus $n = 3$ and $m = 1$.
$$\mathbf{Z} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$
- `Z.constraint=as.factor(c(1,2,3))` Each $\mathbf{y}$ time series corresponds to a different hidden state trajectory. The is the default $\mathbf{Z}$ constraint and in this case $n = m$.
$$\mathbf{Z} = \begin{bmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \end{bmatrix}$$
- `Z.constraint=as.factor(c(1,1,2))` The first two $\mathbf{y}$ time series corresponds to one hidden state trajectory and the third $\mathbf{y}$ time series corresponds to a different hidden state trajectory. Here $n = 3$ and $m = 2$.
$$\mathbf{Z} = \begin{bmatrix} 1\,0 \\ 1\,0 \\ 0\,1 \end{bmatrix}$$
  The $\mathbf{Z}$ constraint can be specified using either numeric or character factor levels. `c(1,1,2)` is the same as `c("north","north","south")`
- `Z.constraint="identity"` This is the default behavior. This means $\mathbf{Z}$ is a $n \times n$ identity matrix and $m = n$. If $n = 3$, it is the same as `Z.constraint=as.factor(c(1,2,3))`.
- `Z.constraint=matrix(..., nrow=n, ncol=m))` Passing in a $n \times m$ matrix, means that $\mathbf{Z}$ is fixed to the values in the matrix. The matrix must be numeric.

4.3.2 A constraint

In the PopMARSS model, $\mathbf{A}$ is a scaling factor that is not user controllable. Only `A.constraint="scaling"` is allowed.

4.3.3 R constraint

The **R** constraint is completely analogous to the **Q** constraint, except that it is $n \times n$ instead of $m \times m$ and its allowable constraints are affected by the presence of missing data points in $\mathbf{y}$. If data are missing, then **R** must be diagonal.

5

Algorithms used in the MARSS package

5.1 Kalman filter and smoother

The MARSS model is a linear dynamical system with discrete time and Gaussian errors. In 1960, Rudolf Kalman published the Kalman filter (Kalman, 1960), a recursive algorithm that solves for the expected value of the hidden state(s) at time t conditioned on the data up to time t: $E(\mathbf{X}_t|\mathbf{Y}_1^t)$. The Kalman filter gives the optimal (lowest mean square error) estimate of the unobserved $\mathbf{x}_t$ based on the observed data up to time t for this class of linear dynamical system. The Kalman smoother (Rauch et al., 1965) solves for the expected value of the hidden state(s) conditioned on all the data: $E(\mathbf{X}_t|\mathbf{Y}_1^T)$. If the errors in the stochastic process are Gaussian, then the estimators from the Kalman filter and smoother are also the maximum-likelihood estimates.

However, even if the the errors are not Gaussian, the estimators are optimal in the sense that they are estimators with the least variability possible. This robustness is one reason the standard Kalman filter is so powerful – it provides well-behaving estimates of the hidden states for all kinds of multivariate autoregressive processes, not just Gaussian processes. The Kalman filter and smoother are widely used in time-series analysis, and there are many textbooks covering it and its applications. In the interest of giving the reader a single point of reference, we use Shumway and Stoffer (2006) as our reference and adopt their notation (for the most part).

The `MARSSkf` function provides the following Kalman filter and smoother outputs:

`xtt1` The expected value of $\mathbf{X}_t$ conditioned on the data up to time $t-1$.
`xtt` The expected value of $\mathbf{X}_t$ conditioned on the data up to time t.
`xtT` The expected value of $\mathbf{X}_t$ conditioned on all the data from time 1 to T. This the smoothed state estimate.
`Vtt1` The variance of $\mathbf{X}_t$ conditioned on the data up to time $t-1$. Denoted P_t^{t-1} in section 4.2 in Shumway and Stoffer (2006).

`Vtt` The variance of $\mathbf{X}_t$ conditioned on the data up to time t. Denoted P_t^t in section 4.2 in Shumway and Stoffer (2006).

`VtT` The variance of $\mathbf{X}_t$ conditioned on all the data from time 1 to T.

`Vtt1T` The covariance of $\mathbf{X}_t$ and $\mathbf{X}_{t-1}$ conditioned on all the data from time 1 to T.

`Kt` The Kalman gain. This is part of the update equations and relates to the amount `xtt1` is updated by the data at time t to produce `xtt`.

`J` This is similar to the Kalman gain but is part of the Kalman smoother. See equation 4.51 in Shumway and Stoffer (2006).

`Innov` This has the innovations at time t, defined as $\boldsymbol{\eta}_t \equiv \mathbf{y}_t$-$E(\mathbf{Y}_t)$. These are the residuals, the difference between the data and their predicted values. See equation 4.40 in Shumway and Stoffer (2006).

`Sigma` This has the $\mathit{\Sigma}_t$, the variance-covariance matrices for the innovations at time t. This is used for the calculation of confidence intervals, the s.e. on the state estimates and the likelihood. See equation 4.41 in Shumway and Stoffer (2006) for the $\mathit{\Sigma}_t$ calculation.

`logLik` The log likelihood of the data conditioned on the model parameters. See the section below on the likelihood calculation.

5.2 The likelihood

The likelihood of data given a specified MARSS model is part of the output of the `MARSSkf` function. The likelihood computation is based on the innovations form of the likelihood (Schweppe, 1965) and uses the output from the Kalman filter:

$$\log L(\mathit{\Theta}|data) = -\frac{N}{2\log(2\pi)} - \frac{1}{2}\left(\sum_{t=1}^{T}\log|\mathit{\Sigma}_t| + \sum_{t=1}^{T}(\boldsymbol{\eta}_t)^\top \mathit{\Sigma}_t^{-1}\boldsymbol{\eta}_t\right) \tag{5.1}$$

where N is the total number of data points and $|\mathit{\Sigma}_t|$ is the determinant of the innovations variance-covariance matrix. Reference equation 4.67 in Shumway and Stoffer (2006); however there are a few differences between the log likelihood output by `MARSSkf` and that described in Shumway and Stoffer (2006).

The standard likelihood calculation (equation 4.67 in Shumway and Stoffer (2006)) is biased when there are missing values in the data. The missing data modifications discussed in section 4.4 in Shumway and Stoffer (2006) do not correct for this bias. Harvey (1989) discusses at length that the standard formula (equation 4.67 in Shumway and Stoffer (2006)) is an inexact likelihood when there are missing values. The bias is minor if there are few missing values, but it becomes severe as the number of missing values increases. Many ecological datasets may have over 25% missing values and this level of missing values leads to a very biased likelihood if ones uses the inexact formula. Harvey (1989) provides some non-trivial ways to compute the exact likelihood. We use instead the exact likelihood correction for missing values that is presented in section 12.3 in Brockwell and Davis (1991). This solution is straight-forward to implement.

The correction involves the following changes to $\boldsymbol{\eta}_t$ and Σ_t in the equation 5.1. Suppose the value $y_{i,t}$ is missing. First, the corresponding i-th value of $\boldsymbol{\eta}_t$ is set to 0. Second, the i-th diagonal value of Σ_t is set to 1 and the off-diagonal elements on the i-th column and i-th row are set to 0.

5.3 Parameter estimation

5.3.1 Kalman-EM algorithm

In MARSS 1.0, only one method is available for parameter estimation: maximum-likelihood parameters via an Expectation-Maximization (EM) algorithm (function `MARSSkem`). EM algorithms are widely used algorithms that extend maximum-likelihood estimation to cases where there are hidden random variables in a model (Dempster et al., 1977; **?**; Harvey, 1989; Harvey and Shephard, 1993).

The EM algorithm finds the maximum-likelihood estimates of the parameters, $\hat{\Theta}$, in a MARSS model using an iterative process. Starting with an initial set of parameters[1], which we will denote $\hat{\Theta}_1$, an updated parameter set $\hat{\Theta}_2$ is obtaining by finding the Θ that maximizing the expected value of the likelihood over the distribution of the states (X) conditioned on Θ_1"

$$\hat{\Theta}_2 = \arg\max_{\Theta} \quad E_{X|\hat{\Theta}_1}[\log L(\Theta|Y_1^T = y_1^T, X)] \tag{5.2}$$

Then using $\hat{\Theta}_2$, an updated parameter set $\hat{\Theta}_3$ is calculated using equation (5.2). This is repeated until the expected log likelihood stops increasing (or increases less than some set tolerance level).

Implementing this algorithm is actually fairly straight-forward, hence its popularity.

1. Set an initial set of parameters, $\hat{\Theta}_1$
2. E step: using the model for the hidden states ($\mathbf{X}$) and $\hat{\Theta}_1$, calculate the expected values of $\mathbf{X}$ conditioned on all the data y_1^T. Also calculate expected values of any functions of $\mathbf{X}$, $g(\mathbf{X})$, that appear in your expected log likelihood function.
3. M step: put those $E(\mathbf{X}|Y_1^T = y_1^T, \hat{\theta}_1)$ and $E(g(\mathbf{X})|Y_1^T = y_1^T, \hat{\theta}_1)$ into your log likelihood function in place of $\mathbf{X}$ (and $g(\mathbf{X})$) and maximize with respect to Θ. That gives you $\hat{\Theta}_2$
4. Repeat the E and M steps until the log likelihood stops increasing

[1] You can choose these however you wish, however choosing something not too far off from the correct values will make the algorithm go faster.

The EM equations in our algorithm, which we term the Kalman-EM algorithm, are based on Shumway and Stoffer (1982) and Ghahramani and Hinton (1996). Our Kalman-EM algorithm is more involved than that presented in these references because our algorithm is for cases where there are constraints within the parameter matrices (shared values, diagonal structure, block-diagonal structure, ...) and where there are fixed values within the parameter matrices. The appendices of **?** give the full derivation of our EM algorithm, which spans many pages and thus is not presented here.

5.3.2 Known problems with the EM algorithm

The EM algorithm is a hill-climbing algorithm and like all hill-climbing algorithms can get stuck on local maxima. The MARSS package includes a Monte-Carlo initial conditions searcher (function `MARSSmcinit`) based on Biernacki et al. (2003) to minimize this problem. EM algorithms are also known to be much slower than Newton methods. Because we work with ecological data, our datasets are replete with missing values. After struggling and failing to get Newton methods to work on our applications, we switched to using the EM algorithm exclusively. It is slow but it consistently fits the model and finds the maximum-likelihood values. Another R package, the DLM package, also fits MARSS models and uses Newton methods (via R's `optim` function). This is probably faster if it works for your application, however we have not used it since our applications all have missing values and give Newton methods problems. Also be aware that Newton methods do not increase in likelihood at each step and ensuring that they find the global maximum can be tricky; the Monte Carlo initial condition search that works for EM algorithms may not work for Newton algorithms. The MARSS package includes a function to convert marss model objects to dlm model objects so one can easily use both packages.

5.3.3 Future parameter estimation algorithms

The top of our to-do list is to add Bayesian parameter estimation, data-cloning, and restricted maximum-likelihood. We have "research" versions of these but developing versions for public consumption will take considerable time. The DLM package is more focused on Bayesian estimation so that is an option if you are interested in Bayesian fitting.

5.4 Parametric and innovations bootstrapping

Bootstrapping can be used to construct frequentist confidence intervals on the parameter estimates (Stoffer and Wall, 1991) and to compute the small-sample AIC corrector for MARSS models (Cavanaugh and Shumway, 1997); the functions `MARSSparamCIs` and `MARSSaic` do these computations. The `MARSSboot` function does parametric and innovations bootstrapping of MARSS models. The innovations bootstrap essentially bootstraps the residuals after model-fitting (called innovations) and uses the algorithm by Stoffer and Wall (1991). This is a semi-parametric bootstrap since is uses, partially, the maximum-likelihood MARSS model. This algorithm cannot be used if there are missing values in the data (unless all values in a given time step are missing). Also for short time series, it gives biased bootstraps because one cannot resample the first few innovations. The `MARSSboot` also provides a fully parametric bootstrap based on using the maximum-likelihood MARSS model to generate bootstrap data. Our research (**?**) indicates that this provides unbiased bootstrap parameter estimates and works with datasets with missing values. Lastly, `MARSSboot` allows one to bootstrap from a numerically estimated Hessian matrix.

5.5 Simulation and forecasting

The `MARSSsimulate` function simulates from a marss model object using the `mvrnorm` function to produce draws from multivariate normal distributions for each time step. The user must pass in a parameter list (for example from `marssfitted$params` and initial conditions.

5.6 Model selection

The package provides a `MARSSaic` function for computing AIC, AICc and AICb. The latter is a small-sample corrector for MARSS models. The bias problem with AIC and AICc for short time series data are shown in Cavanaugh and Shumway (1997) and **?**. AIC and AICc tend to select overly complex MARSS models when the time series data are short. AICb is a small-sample corrector that is unbiased. The algorithm for AICb for MARSS models is given in Cavanaugh and Shumway (1997). The algorithm in Cavanaugh and Shumway (1997) uses the innovations bootstrap (Stoffer and Wall, 1991), which means it cannot be used when there are missing data. We added a parametric bootstrap option for the AICb computation. This allows one to compute AICb when there are missing data and it provides unbiased AIC even for short time series. See **?** for discussion and testing of parametric AICb for MARSS models.

AICb is comprised of the familiar AIC fit term, $-2\log L$, plus a penalty term that is the mean difference between the log likelihood the data under the bootstraped ML parameter estimates and the log likelihood of the data under the original ML parameter estimate:

$$AICb = -2\log L(\hat{\Theta}|\mathbf{y}) + 2\frac{1}{N_b}\sum_{i=1}^{N_b} -2\log\frac{L(\hat{\Theta}^*(i)|\mathbf{y})}{L(\hat{\Theta}|\mathbf{y})} \tag{5.3}$$

where $\hat{\Theta}$ is the ML parameter set under the original data, $\mathbf{y}_1^T \equiv \mathbf{y}$, $\hat{\Theta}^*(i)$ is a ML parameter set estimated from the i-th bootstrapped data set, $\mathbf{y}^*(i)$, and N_b is the number of bootstrap data sets. It is important to notice that the likelihood in the AICb equation is $L(\hat{\Theta}^*|\mathbf{y})$ not $L(\hat{\Theta}^*|\mathbf{y}^*)$. In other words, we are taking the average of the likelihood of the original data given the bootstrapped parameter sets. See appendices of **?** where the algorithm is given in more detail.

6 The MARSS case studies: instructions

The case studies walk you through some analyses of multivatiate population count data using MARSS models and the `PopMARSS` function. This will take you through both the conceptual step (with pencil and paper) and a *R* step which translates that conceptual model into code. The case studies are written as a tutorial. They assume you have downloaded the case study scripts and data sets and are working through the examples.

6.1 Set-up

- Open up R and change your working directory to the folder where the case study data files and scripts are kept. From the command line (the >), the command looks like `setwd("your_directory/MARSS_Case_Studies")` or you can use the *R* GUI (File:Change dir...).
- Type in `dir()` and you should see a list of the case study data files.
- If you haven't already, install the MARSS package. Type from the command line: (Windows) `install.packages("MARSS.zip", repos = NULL)` or (Mac/Unix) `install.packages("MARSS.tar.gz", repos = NULL)`. Mac users need Xtools installed for this to work. You will need write permissions for your *R* program directories to install packages. See the help pages on CRAN for workarounds if you don't have write permission.
- Type in `library(MARSS)` at the *R* command line. Now you should be ready.
- Each case study comes with an associated script file: `Case_Study_#.r` with the code you need to do the basic analyses in the worksheets. It also contains pointers for doing extensions of the basic analyses.

6.2 Tips

- `MARSSprint(foo)` will print the structure of the MARSS model that was fit in the call `foo = PopMARSS(logdata)`. This allows you to double check the model you fit.
- When you run `PopMARSS`, it will output the number of iterations used. If you reached the maximum, re run with `EMOptions=list(max.EMiter=...)` set higher than the default (5000).
- If you misspecify the model, `PopMARSS` will post an error that should give you an idea of the problem. Remember, the number of rows in your data is n, time is across the columns, and the maximum number in `constraint$Z` is m, the number of x time series in your model.
- If you are fitting to population counts, your data must be logged (base e) before being passed in. The default missing value indicator is -99. You can change that by passing in `miss.value=...`.
- Running `PopMARSS` with no arguments except your data will fit an unconstrained MSSM with $m = n$ and a diagonal $\mathbf{R}$ matrix.

7

Case Study 1: Count-based PVA for data with observation error

7.1 The Problem

Estimates of extinction and quasi-extinction risk are an important risk metric used in the management and conservation of endangered and threatened species. By necessity, these estimates are based on data that contain both variability due to real year-to-year changes in the population growth rate (process errors) and variability in the relationship between the true population size and the actual count (observation errors). Classic approaches to extinction risk assume the data have only process error, i.e. no observation error. In reality, observation error is ubiquitous both because of the sampling variability and also because of year-to-year (and day-to-day) variability in sightability.

In this case study, we are use a Kalman filter to fit a univariate (meaning one time series) state-space model to count data for a population. We will compute the extinction risk metrics given in Dennis et al. (1991), however instead of using a process-error only model (as is done in the original paper), we use a model with both process and observation error. The risk metrics and their interpretations are the same as in Dennis et al. (1991). The only real difference is how we compute σ^2, the process error variance. However this difference has a large effect on our risk estimates, as you will see.

In this case study, we use a density-independent model, which is the same as the Gompertz model (4.1) with $\mathbf{B} = 1$. Density-independence is often a reasonable assumption when doing a PVA because we do such calculations for at-risk populations that are either declining or that are well below historical levels (and presumably carrying capacity). In an actual PVA, it is necessary to justify this assumption and if there is reason to doubt the assumption, one tests for density-dependence (Taper and Dennis, 1994) and does sensitivity analyses using state-space models with density-dependence (Dennis et al., 2006).

The univariate model is written:

$$x_t = x_{t-1} + u + e_t \qquad \text{where } e_t \sim Norm(0, \sigma^2) \tag{7.1}$$
$$y_t = x_t + \epsilon_t \qquad \text{where } \epsilon_t \sim Norm(0, \eta^2) \tag{7.2}$$

where y_t is the logarithm of the observed population size at time t, x_t is the unobserved state at time t, u is the growth rate, and σ^2 and η^2 are the process and observation error variances, respectively. In the R code to follow, σ^2 is denoted `Q` and η^2 is denoted `R` (because the functions we are using are also for multivariate state-space models and those models use **Q** and **R** for the respective variance-covariance matrices).

7.2 Simulated data with process and observation error

We'll start by using simulated data to see the difference between data and estimates from a model with process error only versus a model that also includes observation error. For our simulated data, we'll used a decline of 5% per year, process variability of 0.01 (typical for big mammals), and a observation variability of 0.05 (which is a bit on the high end). We'll randomly set 10% of the values as missing. Here's the code:

Set things up.

```
sim.u = -0.05 # growth rate
sim.Q = 0.01 # process error variance
sim.R = 0.05 # non-process error variance
nYr= 30 # number of years of data to generate
fracmissing = 0.1 # fraction of years that are missing
init = 7 # log of initial pop abundance (~1100 individuals)
years = seq(1:nYr) # sequence 1 to nYr
x = rep(NA,nYr) # replicate NA nYr times
y = rep(NA,nYr)
```

First generate the population sizes using equation 7.1:

```
x[1]=init
for(t in 2:nYr)
   x[t] = x[t-1]+ sim.u + rnorm(1, mean=0, sd=sqrt(sim.Q))
```

Add observation error and missing values to generate the observed data using equation 7.2:

```
for(t in 1:nYr)
   y[t]= x[t] + rnorm(1,mean=0,sd=sqrt(sim.R))
missYears =
    sample(years[2:(nYr-1)],floor(fracmissing*nYr),replace = F)
y[missYears]=-99
```

Now let's look at the simulated data. Stochastic population trajectories show much variation, so it is best to look at a few at once. In figure 7.1, nine simulations from the identical parameters (above) are shown.

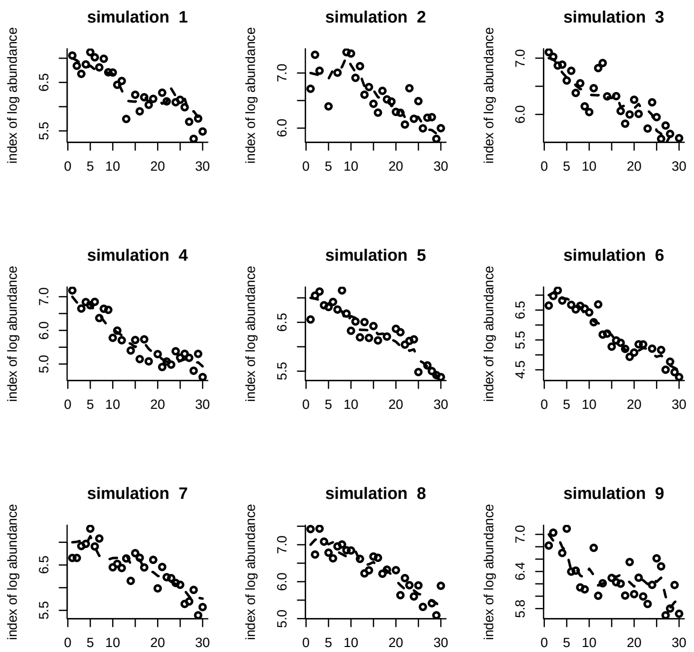

Fig. 7.1. Plot of nine simulated population time series with process and observation error. Circles are observation and the dashed line is the true population size.

Example 7.1 (Look at the effect of parameter values on parameter estimates)

A good way to get a feel for reasonable σ^2 values is to generate simulated data and look at the time series. As a biologist, you probably have a pretty good idea of what kind of year-to-year population changes are reasonable for your species. For example for most of the mammalian species I work with, the maximum population yearly increase would be around 50% (the population could go from 1000 to 1500 in one year), but some of the fish species could easily double or even triple in a really good year. Your observed data may bounce around a lot for many different reasons having to do with sightability, sampling error, age-structure, etc., but the underlying population trajectory is constrained by the kinds of year-to-year changes in population size that are biologically possible for your species. σ^2 describes those true population changes.

Run the Exercise 1 code several times using different parameter values to get a feel for how different the time series can look based on identical parameter *values. You can cut and paste from the pdf into the R command line. Typical vertebrate σ^2 values are 0.002 to 0.02, and typical η^2 values are 0.005 to 0.1. A u of -0.01 translates to an average 1% per year decline and a u of -0.1 translates to an average 10% per year decline (approximately).*

Example 7.1 code

Type `show.doc(MARSS, Case_study_1.R)` to open a file in R with all the example code.

```
par(mfrow = c(3, 3))
sim.u = -0.05
sim.Q = 0.01
sim.R = 0.05
nYr = 30
fracmiss = 0.1
init = 7
years = seq(1:nYr)
for (i in 1:9) {
    x = rep(NA, nYr)
    y = rep(NA, nYr)
    x[1] = init
    for (t in 2:nYr) x[t] = x[t - 1] + sim.u + rnorm(1, mean = 0,
        sd = sqrt(sim.Q))
    for (t in 1:nYr) y[t] = x[t] + rnorm(1, mean = 0, sd = sqrt(sim.R))
    missYears = sample(years[2:(nYr - 1)], floor(fracmiss * nYr),
        replace = FALSE)
    y[missYears] = -99
    plot(years[y != -99], y[y != -99], xlab = "", ylab = "log abundance",
        lwd = 2, bty = "l")
    lines(years, x, type = "l", lwd = 2, lty = 2)
    title(paste("simulation ", i))
}
legend("topright", c("Observed", "True"), lty = c(-1, 2), pch = c(1,
    -1))
```

7.3 Parameter estimation

7.3.1 Maximum-likelihood estimates for a model with observation error

We put the simulated data through the Kalman-EM algorithm in order to estimate the parameters, u, σ^2, and η^2, and population sizes. These are the estimates using a model with process and observation variability. The function call is `kem = KalmanEM(data)`, where `data` is a vector of logged (base e) counts with missing values denoted by -99. After this call, the ML parameter estimates are `kem$U`, `kem$Q` and `kem$R`. There are numerous other outputs from the `KalmanEM` function. To get a list of the outputs type in `names(kem)`. Note that `kem` is just a name; I could have called the output `foo`. Here's some code to fit to the simulated time series.

```
kem = PopMARSS(y)
```

Let's look at the parameter estimates for the nine simulated time series in figure 7.1 to get a feel for the variation. I used the `KalmanEM` function on each time series to produce parameter estimate for each simulation. The estimates are followed by the mean (over the nine simulations) and the true values:

```
kem.params
```

```
               kem.U        kem.Q      kem.R
sim 1    -0.05042596 0.0163014312 0.02726932
sim 2    -0.03546988 0.0094046536 0.06080203
sim 3    -0.05082006 0.0129842111 0.03678076
sim 4    -0.07755399 0.0136617392 0.05310581
sim 5    -0.05138852 0.0081869163 0.03382020
sim 6    -0.08803288 0.0185930371 0.04307711
sim 7    -0.04287942 0.0122679453 0.03865885
sim 8    -0.06358920 0.0008869761 0.06558340
sim 9    -0.03078476 0.0034198589 0.07451882
mean sim -0.05454941 0.0106340854 0.04817959
true     -0.05000000 0.0100000000 0.05000000
```

As expected, the estimate parameters do not exactly match the true parameters, but the average should be fairly close (although 9 simulations is a small sample size). Also note that although we don't get u quite right, our estimates are usually negative. Thus our estimates usually indicate declining dynamics.

The Kalman-EM algorithm also gives an estimate of the true population size with observation error removed. This is in `kem$states`. Figure 7.2 shows the `KalmanEM` estimated true states of the population over time as a solid line. Note that the solid line is considerably closer to the actual true states (dashed line) than the observations. On the other hand with certain datasets, the Kalman filter can get it quite wrong as well!

7.3.2 Maximum-likelihood estimates for a model with no observation error

We used the Kalman-EM algorithm to estimate the mean population rate u and process variability σ^2 under the assumption that the count data have observation error. However, the classic approach to this problem, referred to

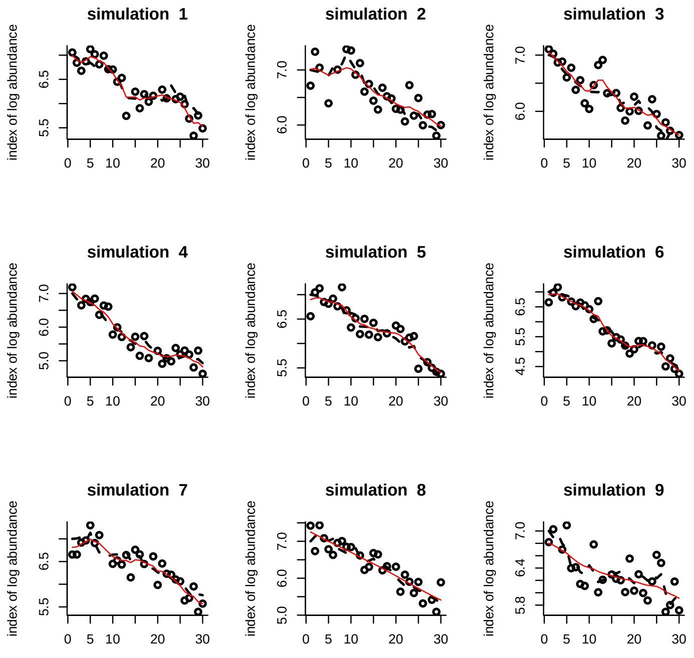

Fig. 7.2. The circles are the observed population sizes with error. The dashed lines are the true population sizes. The solid thin lines are the estimates of the true population size from the Kalman-EM algorithm

as the "Dennis model" (Dennis et al., 1991), uses a model that assumes the data have no observation error; all the variability in the data is assumed to result from process error. This approach works fine if the observation error in the data is low, but not so well if the observation error is high. We will next fit the data using the classic approach so that we can compare and contrast parameter estimates from the different methods.

Using the estimation method in (Dennis et al., 1991), our data need to be re-specified as the observed population changes (`delta.pop`) between censuses along with the time between censuses (`tau`). We re-specify the data as follows:

```
den.years = years[y!=-99] # the non missing years
den.y = y[y!=-99] # the non missing counts
den.n.y = length(den.years)
delta.pop = rep(NA, den.n.y-1 ) # population transitions
tau = rep(NA, den.n.y-1 ) # step sizes
for (i in 2:den.n.y ){
        delta.pop[i-1] = den.y[i] - den.y[i-1]
        tau[i-1] =  den.years[i]-den.years[i-1]
        } # end i loop
```

Next, we regress the changes in population size between censuses (`delta.pop`) on the time between censuses (`tau`) while setting the regression intercept to 0. The slope of the resulting regression line is an estimate of u, while the variance of the residuals around the line is an estimate of σ^2. The regression is shown in Figure 7.3. Here is the code to do that regression:

```
den91 <- lm(delta.pop ~ -1 + tau)
# note: the "-1" specifies no intercept
den91.u = den91$coefficients
den91.Q = var(resid(den91))
```

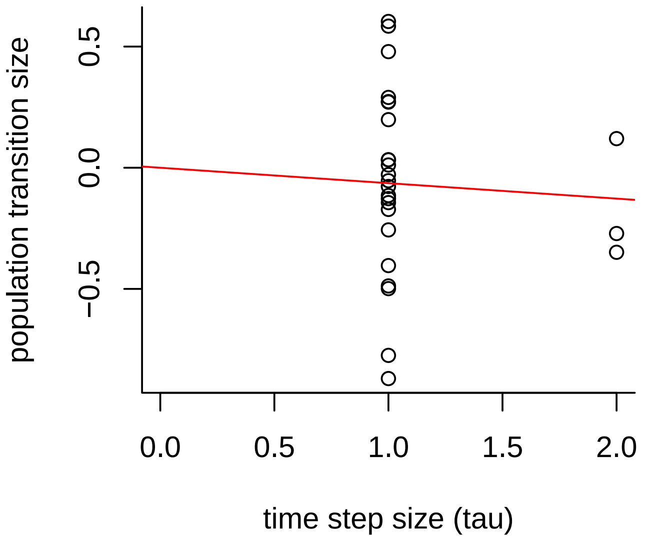

Fig. 7.3. The regression of $log(N_{t+\tau}) - log(N_t)$ against τ. The slope is the estimate of u and the variance of the residuals is the estimate of Q.

Here are the parameters values for the data in figure 7.2 using the process-error only model:

```
den91.params
```

```
             den91.U    den91.Q
sim 1    -0.02739147 0.08007299
sim 2    -0.01083183 0.14451601
sim 3    -0.05314250 0.07740102
sim 4    -0.09150592 0.15158124
sim 5    -0.02300056 0.07665973
sim 6    -0.07777502 0.10946816
sim 7    -0.05895568 0.10631600
sim 8    -0.04797177 0.15152825
sim 9    -0.01926466 0.16075869
mean sim -0.04553771 0.11758912
true     -0.05000000 0.01000000
```

Notice that the u estimates are similar to those from the Kalman-EM algorithm, but the σ^2 estimate (`Q`) is much larger. That is because this approach treats all the variance as process variance, so any observation variance in the data is lumped into process variance (in fact it appears as $2 \times$ the observation variance).

Example 7.2 (Look at the variability in parameter estimates)

In this example, you'll look at how variable the parameter estimates are by generating multiple (`nsim`) simulated data sets and then estimating parameter values for each. You'll compare the Kalman-EM estimates to the estimates using a process error only model (i.e. ignoring the observation error).

Example 7.2 code

Type `show.doc(MARSS, Case_study_1.R)` *to open a file in R with all the example code.*

```
sim.u = -0.05
sim.Q = 0.01
sim.R = 0.05
nYr = 30
fracmiss = 0.1
init = 7
nsim = 9
years = seq(1:nYr)
params = matrix(NA, nrow = 11, ncol = 5, dimnames = list(c(paste("sim",
    1:9), "mean sim", "true"), c("kem.U", "den91.U", "kem.Q",
    "kem.R", "den91.Q")))
for (i in 1:nsim) {
    x.ts = matrix(NA, nrow = nsim, ncol = nYr)
    y.ts = matrix(NA, nrow = nsim, ncol = nYr)
    x.ts[i, 1] = init
    for (t in 2:nYr) x.ts[i, t] = x.ts[i, t - 1] + sim.u + rnorm(1,
        mean = 0, sd = sqrt(sim.Q))
    for (t in 1:nYr) y.ts[i, t] = x.ts[i, t] + rnorm(1, mean = 0,
        sd = sqrt(sim.R))
    missYears = sample(years[2:(nYr - 1)], floor(fracmiss * nYr),
        replace = FALSE)
    y.ts[i, missYears] = -99
    kem = PopMARSS(y.ts[i, ], silent = TRUE)
    params[i, c(1, 3, 4)] = c(kem$par$U, kem$par$Q, kem$par$R)
    den.years = years[y.ts[i, ] != -99]
    den.yts = y.ts[i, y.ts[i, ] != -99]
    den.n.yts = length(den.years)
    delta.pop = rep(NA, den.n.yts - 1)
    tau = rep(NA, den.n.yts - 1)
    for (t in 2:den.n.yts) {
        delta.pop[t - 1] = den.yts[t] - den.yts[t - 1]
        tau[t - 1] = den.years[t] - den.years[t - 1]
    }
    den91 <- lm(delta.pop ~ -1 + tau)
    params[i, c(2, 5)] = c(den91$coefficients, var(resid(den91)))
}
params[nsim + 1, ] = apply(params[1:nsim, ], 2, mean)
params[nsim + 2, ] = c(sim.u, sim.u, sim.Q, sim.R, sim.Q)
```

Here is an example of the output from the code:

```
print(params,digits=3)
```

```
            kem.U den91.U     kem.Q  kem.R den91.Q
sim 1    -0.0766 -0.0755 0.003766 0.0440  0.1081
sim 2    -0.0762 -0.0959 0.025407 0.0252  0.0762
sim 3    -0.0537 -0.0508 0.001003 0.0722  0.1698
sim 4    -0.0451 -0.0687 0.003680 0.0724  0.1753
sim 5    -0.0429 -0.0197 0.019543 0.0229  0.0682
sim 6    -0.0279 -0.0269 0.005187 0.0512  0.1479
sim 7    -0.0420 -0.0442 0.029751 0.0297  0.0859
sim 8    -0.0942 -0.1086 0.009161 0.0332  0.0722
sim 9    -0.0633 -0.0579 0.000832 0.0428  0.0978
mean sim -0.0580 -0.0609 0.010926 0.0437  0.1113
true     -0.0500 -0.0500 0.010000 0.0500  0.0100
```

1. *Re-run the code a few times to see the performance of the estimates using a state-space model (*`kem.`*) versus the model with no observation error (*`den91`*). You can cut and paste the code from the pdf file into an R script file or on to the R command line.*

2. *Alter the observation variance,* `sim.R` *in the data generation step in order to get a feel for performance as observations are further corrupted. What happens as error is increased?*

3. *Decrease the number of years of data,* `nYr` *and re-run the parameter estimation. What changes?*

If you find that the exercise code takes too long to run, reduce the number of simulations (by reducing `nsim` *in the code).*

7.4 Probability of hitting a threshold $\Pi(x_d, t_e)$

A common extinction risk metric is 'the probability that a population will hit a certain threshold x_d within a certain time frame t_e – *if the observed trends continue*'. Under this definition, we can compute $\Pi(x_d, t_e)$ using the stochastic population model (equation 7.1) and our estimate of the parameters of that model. In practice, the threshold used is not $N_e = 1$, which would be true extinction. Often a 'functional' extinction threshold will be used ($N_e >> 1$). Other times a threshold of 'a p_d fraction of current levels' is used. The latter is used because we often have imprecise information about the relationship between the true population size and what we measure in the field; many population counts are index counts. In these cases, one must use 'fractional declines' as the threshold. Also, extinction estimates that use an absolute threshold (like 100 individuals) are quite sensitive to error in the estimate of true population size. In this workshop, we are going to use fractional declines as the threshold, specifically $p_d = 0.1$ which means a 90% decline below the population size at the last census.

$\Pi(x_d, t_e)$ is typically presented as a curve showing the probabilities of hitting the threshold (y-axis) over different time horizons (t_e) on the x-axis. Extinction probabilities can be computed through Monte Carlo simulations or analytically using equation 16 in Dennis et al. (1991) (note there is a typo in equation 16; the last + is supposed to be -). We will use the latter method:

$$\Pi(x_d, t_e) = \pi(u) \times \Phi\left(\frac{-x_d + |u|t_e}{\sqrt{\sigma^2 t_e}}\right) + \exp(2x_d|u|/\sigma^2)\Phi\left(\frac{-x_d - |u|t_e}{\sqrt{\sigma^2 t_e}}\right) \quad (7.3)$$

where x_e is the threshold and is defined as $x_e = log(N_0/N_e)$, where N_0 is the current population estimate and N_e is the threshold. If we are using fractional declines then $x_e = log(N_0/(p_d \times N_0)) = -log(p_d)$. $\pi(u)$ is the probability that the threshold is eventually hit (by $t_e = \infty$). $\pi(u) = 1$ if $u <= 0$ and $\pi(u) = \exp(-2ux_d/\sigma^2)$ if $u > 0$. $\Phi()$ is the cumulative probability distribution of the standard normal (mean = 0, sd = 1). Here is the R code for that computation (using a fractional decline threshold):

```
pd = 0.1 #means a 90 percent decline
tyrs = 1:100
xd = -log(pd)
p.ever = ifelse(u<=0,1,exp(-2*u*xd/Q)) #Q=sigma2
for (i in 1:100){
  Pi[i] = p.ever * pnorm((-xd+abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))
  + exp(2*xd*abs(u)/Q) *
  pnorm((-xd - abs(u)* tyrs[i])/sqrt(Q*tyrs[i]))
       }
```

Figure 7.4 shows the estimated probabilities of hitting the 90% decline for the nine 30-year times series simulated with $u = -0.05$, $\sigma^2 = 0.01$ and $\eta^2 = 0.05$. The dashed line shows the estimates using the Kalman-EM parameter estimates and the solid line shows the estimates using a process-error only model (the `den91` estimates). The circles are the true probabilities. The difference between the estimates and the true probalities is due to errors in $\hat{u}$. Those errors are due largely to process error – not observation error. As we saw earlier, by chance population trajectories with a $u < 0$ will increase, even over a 30-year period. In this case, $\hat{u}$ will be positive when in fact $u < 0$.

Looking at the figure, it is obvious that the probability estimates are highly variable. However, look at the first panel. This is the average estimate (over 9 simulations). Note that on average (over 9 simulations), the estimates are good. If we had averaged over 1000 simulations instead of 9, you would see that the Kalman-EM line falls on the true line. It is an unbiased predictor. While that may seem a small consolation if estimates for individual simulations are all over the map, it is important for correctly specifying our uncertainty about our estimates. Second, rather than focusing on how the estimates and true lines match up, see if there are any forecasts that seem better than others. For example, are 20-year predictions better than 50 and are 100-yr better or worse. In Exercise 3, you'll remake this with different u. You'll discover from that that populations in the worst shape (smallest u) have better predictions.

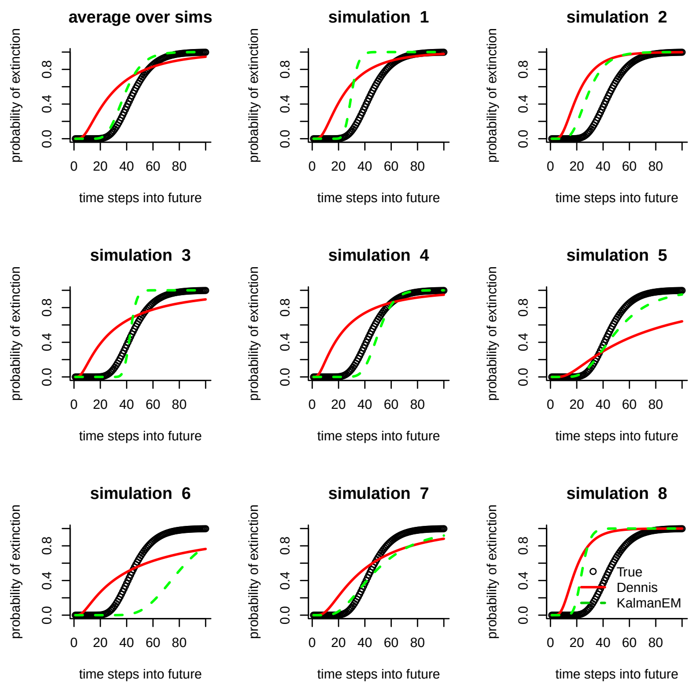

Fig. 7.4. Plot of the true and estimated probability of declining 90% in different time horizons (the x axis) for nine simulated population time series with observation error.

Example 7.3 (The effect of parameter values on risk estimates)

In this example, you'll recreate figure 7.4 using different parameter values. This will give you a feel for how variability in the data and population pro-

cess affect the risk estimates. You'll need to run the Example 7.2 code before running the Example 7.3 code.

Example 7.3 code

Type `show.doc(MARSS, Case_study_1.R)` to open a file with all the example code.

```
par(mfrow = c(3, 3))
pd = 0.1
te = 100
tyrs = 1:te
xd = -log(pd)
for (j in c(10, 1:8)) {
    real.ex = matrix(nrow = te)
    denn.ex = matrix(nrow = te)
    kal.ex = matrix(nrow = te)
    u = params[j, 1]
    Q = params[j, 3]
    p.ever = ifelse(u <= 0, 1, exp(-2 * u * xd/Q))
    for (i in 1:100) {
        kal.ex[i] = p.ever * pnorm((-xd + abs(u) * tyrs[i])/(sqrt(Q) *
            sqrt(tyrs[i]))) + exp(2 * xd * abs(u)/Q) * pnorm((-xd -
            abs(u) * tyrs[i])/(sqrt(Q) * sqrt(tyrs[i])))
    }
    u = params[j, 2]
    Q = params[j, 5]
    p.ever = ifelse(u <= 0, 1, exp(-2 * u * xd/Q))
    for (i in 1:100) {
        denn.ex[i] = p.ever * pnorm((-xd + abs(u) * tyrs[i])/(sqrt(Q) *
            sqrt(tyrs[i]))) + exp(2 * xd * abs(u)/Q) * pnorm((-xd -
            abs(u) * tyrs[i])/(sqrt(Q) * sqrt(tyrs[i])))
    }
    u = sim.u
    Q = sim.Q
    p.ever = ifelse(u <= 0, 1, exp(-2 * u * xd/Q))
    for (i in 1:100) {
        real.ex[i] = p.ever * pnorm((-xd + abs(u) * tyrs[i])/sqrt(Q *
            tyrs[i])) + exp(2 * xd * abs(u)/Q) * pnorm((-xd -
            abs(u) * tyrs[i])/sqrt(Q * tyrs[i]))
    }
    plot(tyrs, real.ex, xlab = "time steps into future", ylab = "probability of extinction
        ylim = c(0, 1), bty = "l")
    if (j <= 8)
        title(paste("simulation ", j))
    if (j == 10)
        title("average over sims")
    lines(tyrs, denn.ex, type = "l", col = "red", lwd = 2, lty = 1)
    lines(tyrs, kal.ex, type = "l", col = "green", lwd = 2, lty = 2)
}
legend("bottomright", c("True", "Dennis", "KalmanEM"), pch = c(1,
    -1, -1), col = c(1, 2, 3), lty = c(-1, 1, 2), lwd = c(-1,
    2, 2), bty = "n")
```

1. *Change* `sim.R` *and rerun the Example 7.2 code. Then run the Example 7.3 code. When are the estimates using the process-error only model (*`den91`*) worse and in what way are they worse?*

2. *You might imagine that you should always use a model that assumes that the data contain observation error, since in practice observations are never perfect. However, there is a cost to estimating that extra variance parameter and the cost is a more variable σ^2 (*`Q`*) estimate. Play with shortening the time series and decreasing the* `sim.R` *values. Are there situations when the 'cost' of the extra parameter is greater than the 'cost' of ignoring observation error?*

3. *How does changing the extinction threshold (*`pd`*) change the extinction probability curves? (Do not remake the data, i.e. don't rerun the Example 7.2 code.)*

4. *How does changing the rate of decline (*`sim.u`*) change the estimates of risk? Rerun the Example 7.2 code using a lower u; this will create a new matrix of parameter estimates. Then run the Example 7.3 code. Do the estimates seem better of worse for rapidly declining populations?*

5. *Rerun the Example 7.2 code using fewer number of years (*`nYr` *smaller) and increase* `fracmissing`*. Then run the Example 7.3 code. The graphs will start to look peculiar. Why do you think it is doing that? Hint: look at the estimated parameters.*

7.5 Certain and uncertain regions

From exercise 3, you've observed one of the problems with estimates of the probability of hitting thresholds. Looking over the 9 simulations, your risk estimates will be on the true line sometimes and other times they are way off. So your estimates are variable. Using only the point estimates of the probability of 90% decline by themselves in a PVA should not be done. At the minimum, CIs need to be added (next section), but even with CIs, the probability of hitting declines often doesn't capture our certainty and uncertainty about our risk estimates.

From exercise 3, you might have also noticed that there are some time horizons (10, 20 years) for which the estimate are highly certain (not hitting the threshold), while for other time horizons (30, 50 years) the estimates are all over the map. Put another way, you may be able to say with high confidence that a 90% decline will NOT occur between years 1 to 20 and that by year 100 it most surely will have occurred. However, between the years 20 and 100, you are very uncertain about the risk. The point is that you can be certain about some forecasts while at the same time being uncertain about other forecasts.

One way to show this is to plot the uncertainty as a function of the forecast, where the forecast is defined in terms of the forecast length (number of years) and forecasted decline (percentage). Uncertainty is defined as how much of the 0-1 range your 95% CI covers. Ellner and Holmes (2008) show such a figure (their figure 1). Figure 7.5 shows a version of this figure that you can produce with the function `CSEGtmufigure(u= val, N= val, s2p= val)`. In the figure, I used $u = -0.05$ which is a 5% per year decline, $N = 25$ so 25 years between the first and last census, and $s_p^2 = 0.01$. The process variability for big mammals is typically in the range of 0.002 to 0.02.

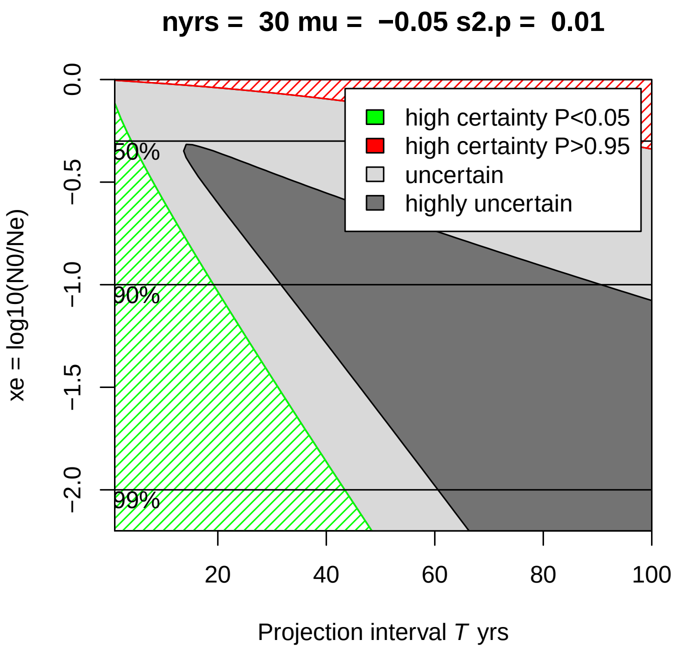

Fig. 7.5. This figure shows your region of high uncertainty (dark grey). In this region, the minimum 95% CIs (meaning if you had no observation error) span 80% of the 0 to 1 probability. That is, you are uncertain if the probability of a specified decline is close to 0 or close to 1. The green (dots) shows where your upper 95% CIs does not exceed P=0.05. So you are quite sure the probability of a specified decline is less than 0.05. The red (dots) shows where your lower 95% CIs is above P=.95. So you are quite sure the probability is greater than P=0.95. The light grey is between these two certain/uncertain extremes.

Example 7.4 (Uncertain and certain regions)

Use the Example 7.4 code to re-create Figure 7.5 and get a feel for when (what parameter ranges) risk estimates are more certain and when they are less certain.

Exercise 7.4 code
Type `show.doc(MARSS, Case_study_1.R)` to open a file in R with all the example code.

```
par(mfrow = c(1, 1))
CSEGtmufigure(N = 30, u = -0.05, s2p = 0.01)
```

`N` *are the number of years of data,* `u` *is the mean population growth rate, and* `s2p` *is the process variance.*

7.6 More risk metrics and some real data

The previous sections have focused on the probability of hitting thresholds because this is an important and common risk metric used in PVA and it appears in IUCN Red List criteria. However, as you have seen, there is high uncertainty associated with such estimates. Part of the problem is that probability is constrained to 0 to 1, and it is easy to get estimates with CIs that span 0 to 1. Other metrics of risk, $\hat{u}$ and the distribution of the time to hit a threshold (Dennis et al., 1991), don't have this problem and may be more informative. Figure 7.6 shows different risk metrics from Dennis et al. (1991) on a single plot. This figure is generated by the call

```
dat=read.table(datafile, skip=1)
dat=as.matrix(dat)
CSEGriskfigure(dat)
```

The `datafile` is the name of the data file, with column 1 = years and column 2 = population count (logged). `CSEGriskfigure()` has a number of arguments that can be passed in to change the default behavior. The variable `te` is the forecast length (default is 100 years), `threshold` is the extinction threshold either as an absolute number, if `absolutethresh=T`, or as a fraction of current population count, if `absolutethresh=F`. The default is `absolutethresh=F` and `threshold=0.1`. `datalogged=TRUE` means the data are already logged; this is the default.

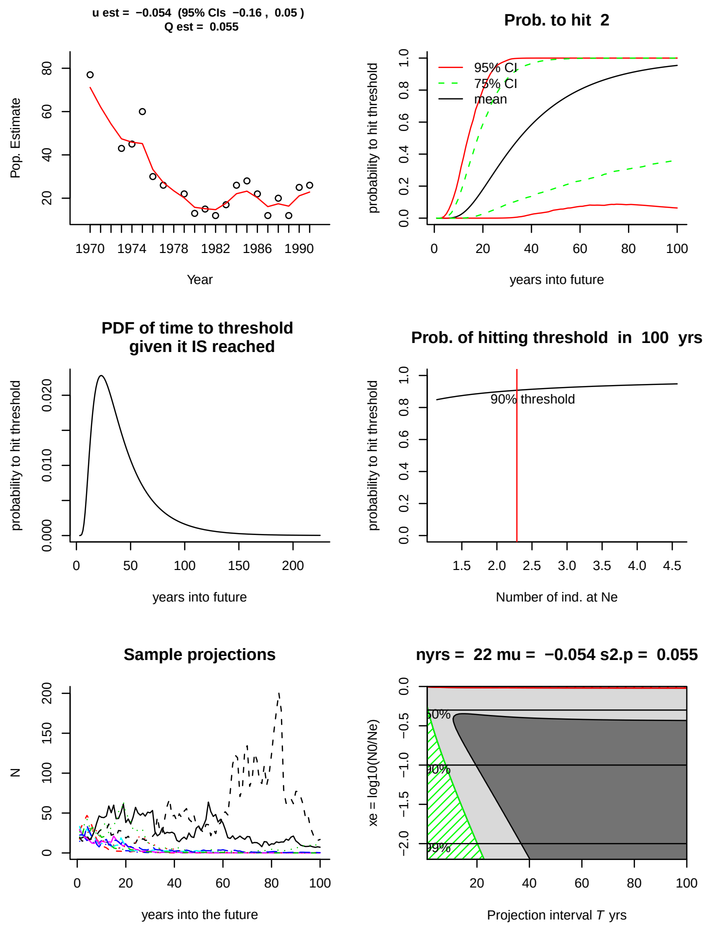

Fig. 7.6. Risk figure using data for the critically endangered African Wild Dog (data from Ginsberg et al. 1995). This population went extinct after 1992.

Example 7.5 (Example of risk figures for different species)

Use the Example 7.5 code to re-create Figure 7.6. The package includes other data for you to run: `prairiechicken` *from the endangered Attwater Prairie Chicken,* `graywhales` *from Gerber et al. (1999), and* `grouse` *from the Sharp-tailed Grouse (a species of U.S. federal concern) in Washington State. If you have other textfiles of data, you can run those too. The commented lines show how to read in data from a tab-delimited text file with a header line.*

Exercise 5 code

Type show.doc(MARSS, Case_study_1.R) to open a file in R with all the example code.

```
#If you have your data in a tab delimited file with a header
#This is how you would read it in using file.choose()
#to call up a directory browser.
#However, the package has the datasets for the exercises
#dat=read.table(file.choose(), skip=1)
#dat=as.matrix(dat)
dat = wilddogs
CSEGriskfigure(dat, CI.method="hessian", silent=TRUE)
```

7.7 Confidence intervals

The figures produced by `CSEGriskfigure()` have confidence (95% and 75%) on the probabilities in the top right panel. The standard way to produce these CIs is via parametric bootstrapping. Here are the steps in a parametric bootstrap:

- You estimate u and σ^2 and η^2
- Then you simulate time series using those estimates and equations 7.1 and 7.2
- Then you re-estimate your parameters from the simulated data (using say `KalmanEM(simdata)`
- Repeat for 1000s of time series simulated using your estimated parameters. This gives you a large set of bootstrapped parameter estimates
- For each bootstrapped parameter set, compute a set of extinction estimates (you use equation 7.3 and code from exercise 3)
- The α% ranges on those bootstrapped extinction estimates gives you your α CIs on your probabilities of hitting thresholds

Look at the code in `CSEGriskfigure.r` to see how to do this in R (type `CSEGriskfigure` at the R command line and the code will be shown).

Producing parameter estimates by estimating them from the simulated data would be quite slow. Therefore for the manual, I used approximate CIs on the parameters using the inverse of a numerically estimated Hessian matrix. This uses an estimate of the variance-covariance matrix of the parameters from the inverse of a numerically estimated Hessian matrix. The function `CSEGriskfigure()` has an option you can set `CI.method = c("hessian", "paramboot", "nonparamboot", "none")` which tells it how to compute the CIs. I set `CI.method="hessian"`. Using an estimated Hessian matrix to compute

CIs is a handy trick that can be used for all sorts of maximum-likelihood parameter estimates. Look at the code in `CSEGriskfigure()` to see how to use the `nlme` package in *R* to do this easily.

7.8 Other parameter estimation methods

Restricted maximum-likelihood algorithms are also available for state-space models, both univariate and multivariate (Staples et al., 2004; Hinrichsen, 2009). REML can give parameter estimates with lower variance than the Kalman-EM algorithm. Also the REML algorithm is much easier to code than the Kalman-EM algorithm (see code provided with the cited papers). However, the algorithms for REML when there are missing values are not currently available, so you are limited to data with no missing values (at the moment). Data with cycles, from age-structure or predator-prey interactions, are difficult to analyze and both REML and Kalman-EM will give poor estimates for this type of data. The slope method (Holmes, 2001), while more ad-hoc, is robust to those problems. Holmes et al. (2007) used the slope method in a large study of data from endangered and threatened species. Ellner and Holmes (2008) showed that the slope estimates are close to the theoretical minimum uncertainty. However estimates using the slope method are not easily extended to multi-site data. If you wish, you can run the slope method on the data in this case study by using the function `slopemethod(logged.data)`; replace `logged.data` with your time series of data. The function will output u, σ^2, and η^2. See the reference list on the workshop website for a bibliography of papers on maximum-likelihood estimation of state-space models for ecological data.

My research is focused on Kalman-based and REML algorithms because of they are true maximum-likelihood methods, and the research I do on model selection requires that. However if I am doing a PVA and have a single time series with fewer than 25 years of data, I will often use the slope method because that method is less data-hungry. I am using the Kalman-based methods in this workshop because they allow one to easily study multi-site data and we don't have to worry about lots of missing values. One reason the EM algorithm is popular is that it is quite simple conceptually and if coded correctly, must increase in likelihood at each iteration. However, the EM algorithm is slower, sometimes much, much slower, than Newton-based methods. For any but the simplest model structures with few missing values, we have not had success getting Newton-based methods to work via the `optim` function in *R*. We have not tried creating a customized Newton method for our problems, in part because we are trying to write code for general model structures and in part, because it seemed hard. However, if you need a very fast algorithm, you should look into the research on Newton methods for state-space models. For our purposes, the Kalman-EM algorithm is fast enough and it is quite robust and likely to work on any data students might bring to our workshops.

Bayesians: Bayesian applications using state-space models to analyze population data are also well developed. See the reference list on the website for a summary of this literature. The MathBio group at Northwest Fisheries Science Center is actively developing and using Bayesian approaches also. You can find links to this code and research at my website:

http://faculty.washington.edu/eeholmes.

8

Case study 2: Combining multi-site and subpopulation data to estimate trends and trajectories

8.1 The problem

In this case study, we will use multivariate state-space models to combine surveys from multiple sites into one estimate of the average long term population growth rate and the year-to-year variability in that growth rate. Note this is not quite the same as estimating the 'trend'; 'trend' often means what population change happened, whereas the long-term population growth rate refers to the underlying population dynamics. We will use as our example a dataset from harbor seals in the Puget Sound, Washington, USA.

We have five regions where harbor seals were censused from 1978-1999 while hauled out of land[1]. During the period of this dataset, harbor seals were recovering steadily after having been reduced to low levels by hunting prior to protection. The methodologies were consistent throughout the 20 years of the data but we do not know what fraction of the population that each region represents nor do we know the observation-error variance for each region. Given differences between behaviors of animals in different regions and the numbers of haul-outs in each region, the observation errors may be quite different. The regions have had different levels of sampling; the best sampled region has only 4 years missing while the worst has over half the years missing.

Figure 8.1 shows the data. The numbers on each line denote the different regions:

```
1 SJF
2 SJI
3 EBays
4 PSnd
5 HC
```

[1] Jeffries et al. 2003. Trends and status of harbor seals in Washington State: 1978-1999. Journal of Wildlife Management 67(1):208–219

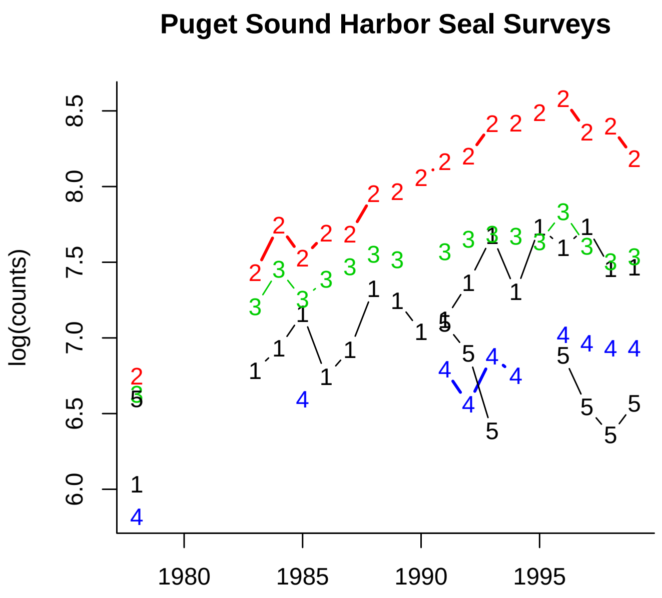

Fig. 8.1. Plot of the of the count data from the five harbor seal regions (Jeffries et al. 2003). Each region is an index of the total harbor seal population, but the bias (the difference between the index and the true population size) for each region is unknown.

For this case study, we will assume that the underlying population process is a stochastic exponential growth process with rates of increase that were not changing through 1978-1999. However, we are not sure if all five regions sample a single "total Puget Sound" population or if there are independent subpopulations. You are going to estimate the long-term population growth rate using different assumptions about the population structures (1 big population versus multiple smaller ones) and observation error structures to see how your assumptions change your estimates.

The data for this case study are in the MARSS package. The data have time running down the rows and we need time across the columns for the `PopMARSS` function, so we will transpose the data:

```
dat = t(harborSealWA)
years = dat[1, ]
n = nrow(dat) - 1
dat = dat[2:nrow(dat), ]
dat <- read.csv("datafile.csv",header=TRUE)
dat <- read.table("datafile.csv",header=TRUE)
```

If you needed to read data in from a comma-delimited or tab-delimited file, these are the commands to do that:

```
dat = t(harborSealWA)
years = dat[1, ]
n = nrow(dat) - 1
dat = dat[2:nrow(dat), ]
```

```
dat <- read.csv("datafile.csv",header=TRUE)
dat <- read.table("datafile.csv",header=TRUE)
```

The years (`years`) are in row 1 of `dat` and the logged data are in the rest of the rows. The number of observation time series (`n`) is the number of rows in `dat` minus 1 (for years row). Let's look at the first few years of data:

```
print(harborSealWA[1:8,], digits=3)
```

```
     Years    SJF    SJI  EBays   PSnd    HC
[1,]  1978   6.03   6.75   6.63   5.82   6.6
[2,]  1979 -99.00 -99.00 -99.00 -99.00 -99.0
[3,]  1980 -99.00 -99.00 -99.00 -99.00 -99.0
[4,]  1981 -99.00 -99.00 -99.00 -99.00 -99.0
[5,]  1982 -99.00 -99.00 -99.00 -99.00 -99.0
[6,]  1983   6.78   7.43   7.21 -99.00 -99.0
[7,]  1984   6.93   7.74   7.45 -99.00 -99.0
[8,]  1985   7.16   7.53   7.26   6.60 -99.0
```

The `-99`'s in the data are missing values. The algorithm will ignore those values when estimating $x_{1:T}$.

8.2 Analyze assuming a single total Puget Sound population

The first step in a state-space modeling analysis is to specify the population structure and how the regions relate to that structure. The general state-space model is

$$\mathbf{X}_t = \mathbf{B}\mathbf{X}_{t-1} + \mathbf{U} + \mathbf{E}_t, \text{ where } \mathbf{E}_t \sim \text{MVN}(0, \mathbf{Q}) \tag{8.1}$$
$$\mathbf{Y}_t = \mathbf{Z}\mathbf{X}_t + \mathbf{A} + \boldsymbol{\eta}_t, \text{ where } \boldsymbol{\eta}_t \sim \text{MVN}(0, \mathbf{R}) \tag{8.2}$$

where all the bolded symbols are matrices. To specify the structure of the population and observations, we will specify what those matrices look like.

8.2.1 A single population process, X

When we are looking at trends over a large geographic region, we might make the assumption that the different census sites are measuring a single population if we think animals are moving sufficiently such that the whole area (multiple regions together) is ”well-mixed”. We write a model of the population abundance as:

$$n_t = exp(u + e_t)n_{t-1}, \tag{8.3}$$

where n_t is the total count in year t, u is the mean population growth rate, and e_t is the deviation from that average in year t. We then take the log of both sides and write the model in log space:

$$x_t = x_{t-1} + u + e_t. \tag{8.4}$$

$x_t = \log n_t$. When there is one effective population, there is one x, there for $\mathbf{X}_t$ is a 1×1 matrix. There is one population growth rate (u) and there is one process variance (σ^2). Thus $\mathbf{U}$ and $\mathbf{Q}$ are 1×1 matrices.

8.2.2 The observation process, Y

For this first analysis, we assume that all five regional time series are observing this one population trajectory but they are scaled up or down relative to that trajectory. In effect, we think that animals are moving around a lot and our regional samples are some fraction of the population. There is year-to-year variation in the fraction in each region, just by chance. Notice that under this analysis, we don't think the regions represent independent subpopulations but rather independent observations of one population.

Our model for the data, $\mathbf{Y}_t = \mathbf{A} + \mathbf{Z}\mathbf{X}_t + \boldsymbol{\eta}_t$, is written out as:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} \epsilon_{1,t} \\ \epsilon_{2,t} \\ \epsilon_{3,t} \\ \epsilon_{4,t} \\ \epsilon_{5,t} \end{bmatrix} \tag{8.5}$$

Each y_i is the time series for a different region (the names for the numbered regions are given on page 2). The A's are the bias between the regional sample and the total population. The A's are scaling (or intercept-like) parameters that are not important for trend estimation[2]. We will ignore them [3]. We

[2] To get rid of the A's, we scale multiple observation time series against each other; thus one A will be fixed at 0

[3] Estimating the bias between regional indices and the total population is important for getting an estimate of the total population size. However, the time series analysis that we are doing for this workshop is not useful for estimating A's. Instead one uses some type of mark-recapture data. For trend estimation, the A's are not important. The regional observation variance captures increased variance due to a regional being a smaller sample of th total population.

allow that each region could have a unique observation variance and that the observation errors are independent between regions. Lastly, we assume that the observations errors on log(counts) are normal and thus the errors on (counts) are log-normal.[4]

We specify independent observation errors with unique variances by $\epsilon_t \sim MVN(0, \mathbf{R})$, where

$$\mathbf{R} = \begin{bmatrix} \eta_{1,t} & 0 & 0 & 0 & 0 \\ 0 & \eta_{2,t} & 0 & 0 & 0 \\ 0 & 0 & \eta_{3,t} & 0 & 0 \\ 0 & 0 & 0 & \eta_{4,t} & 0 \\ 0 & 0 & 0 & 0 & \eta_{5,t} \end{bmatrix} \tag{8.6}$$

$\mathbf{Z}$ is specifying which observation time series, $y_{i,1:T}$, is associated with which population trajectory, $x_{j,1:T}$. $\mathbf{Z}$ is like a look up table with 1 row for each of the n observation time series and 1 column for each of the m population trajectories. A 1 in row i column j means that observation time series i is measuring state process j. Otherwise the value in $\mathbf{Z}_{ij} = 0$. Since we have only 1 population trajectory, all the regions must be measuring that one population trajectory. Thus $\mathbf{Z}$ is $n \times 1$.

8.2.3 Set the constraints for `PopMARSS`

Now that we have specified our state-space model, we set the arguments that will tell the function `PopMARSS` the structure of our model. We do this by passing in the argument `constraint` to `PopMARSS`. `constraint` is a list which specifies any constraints on Z, U, Q, etc. The function call will now look like:

```
kem = PopMARSS(dat, constraint=list(Z=Z.constraint, U=U.constraint,
      Q=Q.constraint, R=R.constraint) )
```

First we set the Z constraint. We need to tell the `PopMARSS` function that $\mathbf{Z}$ is a column vector of 1s (as in equation 8.5). We do this by specifying which data time series belongs to which population using a $1 \times n$ vector *as an object of class factor*. The i-th element specifies which population trajectory the i-th observation time series belongs to. Since there is only one population trajectory in analysis 1, we will have a vector of five 1's. Every observation time series is measuring the first, and only, population trajectory. In later analyses, you'll see how to specify the constraint on Z when we have multiple populations.

```
Z.constraint = as.factor(c(1,1,1,1,1))
```

Note, these are the levels corresponding to each of the n time series and thus must be wrapped in `as.factor()` so that `PopMARSS` recognizes it. You can

[4] The assumption of normality is not unreasonable since these regional counts are the sum of counts across multiple haul-outs.

use either numeric or character levels. Next we specify that the **R** variance-covariance matrix only has terms on the diagonal (the variances) with the off-diagonal terms (the covariances) equal to zero.[5] `constraint`.

```
R.constraint = "diagonal and unequal"
```

The `and unequal` part specifies that the variances are allowed to be unique on the diagonal. If we wanted to force the observation variances to be equal at all regions, we would use `diagonal and equal`. For analysis 1, we only need to set constraints on Z and R. Since there is only one population, there is only one U and Q (they are scalars), so there are no constraints to set on them.

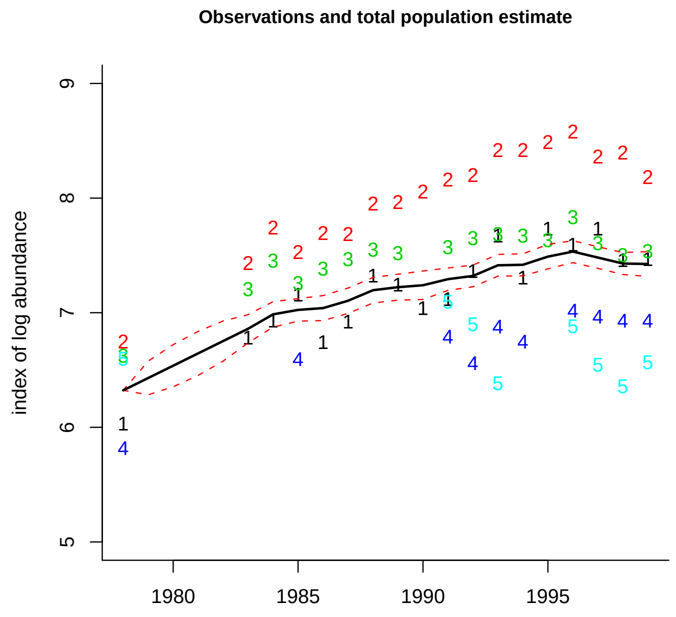

Fig. 8.2. Plot of the estimate of "ln total harbor seals in Puget Sound" (minus the unknown bias for time series 1) against the data. The estimate of the total seal count has been scaled relative to the first time series. The 95% CIs on the population estimates are the dashed lines. These are not the CIs on the observations and the observations (the numbers) should not fall between the CI lines.

[5] For the EM function that we wrote for this workshop, the measurement errors must be uncorrelated if there are missing values in the data.

8.2.4 The `PopMARSS` output

The output from `PopMARSS`, here assigned the name `kem`, is a list of objects. To see all the objects in it:

The following are some of the most used objects. `kem1$states` are the maximum-likelihood estimates of "total harbor seal population" scaled to the first observation data series (Fig. 8.2), and `kem1$states.se` are the standard errors on those estimates. To get 95% CIs, use `kem1$states +/- 1.96*kem1$states.se`. One of the biases, the As, cannot be estimated and arbitrarily our algorithm choses $A_1 = 0$, so the population estimate is scaled to the first observation time series. Since we are only trying to estimate the trend, u, the unknown bias is unimportant. Figure 8.2 shows a plot of `kem1$states` with its 95% CIs over the data. Because `kem1$states` has been scaled relative to the first time series, it is on top of that time series.

The estimated parameters are a list in `kem1`: `kem1$par`. To get the element U of that list, which is the estimated long term population growth rate, type in `kem1$par$U`. Multiply by 100 to get the percent increase per year. The estimated process variance is given by `kem1$par$Q`. The log-likelihood of this model is `kem1$logLik`. We estimated 1 initial x ($t = 0$), 1 process variance, 1 U, 4 A's, and 5 observation variances's. So $K = 12$ parameters. The AIC of this model is $-2 \times loglike + 2K$, which we can show by typing `kem1$AIC`.

Example 8.1 (Fit the single population model to the harbor seal data)

Analyze the harbor seal data using a single population model. The code for Example 8.1 shows you how to input data and send it to the function `PopMARSS`*. When you run* `PopMARSS`*, it will print information on the structure of the model it is fitting and how many iterations it took to run. As you run the examples, add the estimates to the table at the end of the chapter so you can compare estimates across the examples.*

Example 8.1 code

Type `show.doc(MARSS, Case_study_2.R)` *to open a file in R with all the example code.*

```
dat = t(harborSealWA)
years = dat[1, ]
n = nrow(dat) - 1
dat = dat[2:nrow(dat), ]
legendnames = (unlist(dimnames(dat)[1]))
Z.constraint = as.factor(c(1, 1, 1, 1, 1))
R.constraint = "diagonal and unequal"
kem1 = PopMARSS(dat, constraint = list(Z = Z.constraint, R = R.constraint))
matplot(years, t(dat), xlab = "", ylab = "index of log abundance",
    pch = c("1", "2", "3", "4", "5"), ylim = c(5, 9), bty = "L")
lines(years, kem1$states - 1.96 * kem1$states.se, type = "l",
    lwd = 1, lty = 2, col = "red")
lines(years, kem1$states + 1.96 * kem1$states.se, type = "l",
    lwd = 1, lty = 2, col = "red")
lines(years, kem1$states, type = "l", lwd = 2)
title("Observations and total population estimate", cex.main = 0.9)
kem1$par
kem1$logLik
kem1$AIC
```

8.3 Changing the assumption about the observation variances

The variable `kem1$par$R` contains the estimates of the observation error variances. It is a matrix. Here is **R** from Example 8.1:

```
print(kem1$par$R, digits=3)
```

```
         SJF:1  SJI:2 EBays:3 PSnd:4  HC:5
SJF:1   0.0325 0.0000  0.0000 0.0000 0.000
SJI:2   0.0000 0.0355  0.0000 0.0000 0.000
EBays:3 0.0000 0.0000  0.0131 0.0000 0.000
PSnd:4  0.0000 0.0000  0.0000 0.0113 0.000
HC:5    0.0000 0.0000  0.0000 0.0000 0.195
```

Notice that the variances along the diagonal are all different–we estimated 5 unique observation variances. We might be able to improve the fit (relative to the number of estimated parameters) by assuming that the observation variance is equal across regions but the errors are independent. This means we estimate 1 observation variance instead of 5. This is a fairly standard assumption for data that come from the same survey methodology[6].

To impose this constraint, we set the R constraint to

```
R.constraint="diagonal and equal"
```

This tells PopMARSS that all the η^2's along the diagonal in $\mathbf{R}$ are the same. To fit this model to the data, call PopMARSS as:

```
Z.constraint = as.factor(c(1,1,1,1,1))
R.constraint = "diagonal and equal"
kem2 = PopMARSS(dat, constraint=
       list(Z=Z.constraint, R=R.constraint))
```

We estimated 1 initial x, 1 process variance, 1 U, 4 A's, and 1 observation variance. So $K = 8$ parameters. The AIC for this new model compared to the old model with 5 observation variances is:

```
c(kem1$AIC,kem2$AIC)
```

```
[1] -10.231173   8.384659
```

A smaller AIC means a better model. The difference between the 1 observation variance versus the unique observation variances is >10, suggesting that the unique observation variances model is better. Go ahead and type in the R code. Then add the parameter estimates to the table at the back.

One of the key diagnostics when you are comparing fits from multiple models, it to examine whether the model is flexible enough to fit the data. You do this by looking for temporal trends in the the residuals between the estimated population states (e.g. kem2$states) and the data. In Fig. 8.3, the residuals for analysis 2 are shown. Ideally , these residuals should not have a temporal trend. They should look cloud-like. The fact that the residuals for analysis 2 have a strong temporal trend is an indication that our 1 population model is too restrictive for the data[7].

Example 8.2 (Fit a model with shared observation variances)

Analyze the data using the same population model as in example 1, but constrain the $\mathbf{R}$ *matrix so that all sites have the same observation variance. The*

[6] This is not a good assumption for these data since the number haul-outs in each region varies and the regional counts are the sums across all haul-outs in a region. We'll see that this is a poor assumption when we look at the AIC values.

[7] When comparing models via AIC, it is important that you only compare models that are flexible enough to fit the data. Fortunately, inadequate models will usually have very high AICs and fall out of the mix.

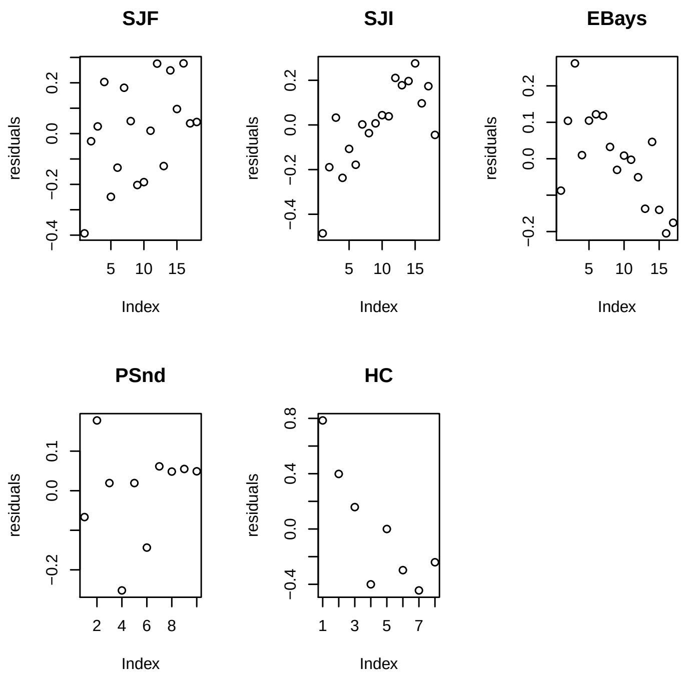

Fig. 8.3. Residuals for the model with a single population. The plots of the residuals should not have trends with time, but they do... This is an indication that the single population model is inconsistent with the data. The code to make this plot is given in the script file for case study 2.

Example 8.2 code shows you how to do this. It also shows you how to make the diagnostics figure (Figure 8.3).

Example 8.2 code

Type `show.doc(MARSS, Case_study_2.R)` to open a file in R with all the example code.

```
#fit model
Z.constraint = as.factor(c(1,1,1,1,1))
R.constraint = "diagonal and equal"
kem2 = PopMARSS(dat, constraint=
       list(Z=Z.constraint, R=R.constraint))
#show parameters
kem2$par$U      #population growth rate
kem2$par$Q      #process variance
kem2$par$R[1,1]  #observation variance
kem2$logLik #log likelihood
c(kem1$AIC,kem2$AIC)
#plot residuals
plotdat = t(dat); plotdat[plotdat == -99] = NA;
matrix.of.biases = matrix(kem2$par$A,
   nrow=nrow(plotdat),ncol=ncol(plotdat),byrow=T)
xs = matrix(kem2$states,
   nrow=dim(plotdat)[1],ncol=dim(plotdat)[2],byrow=F)
resids = plotdat-matrix.of.biases-xs
par(mfrow=c(2,3))
for(i in 1:n){
        plot(resids[!is.na(resids[,i]),i],ylab="residuals")
        title(legendnames[i])
        }
par(mfrow=c(1,1))
```

8.4 Analyze the data assuming North and South subpopulations

For the third analysis, we will change our assumption about the structure of the population. We will assume that there are 2 subpopulations, North and South, and that regions 1 and 2 (Strait of Juan de Fuca and San Juan Islands) fall in the north subpopulation and regions 3, 4 and 5 fall in the south subpopulation. For this analysis, we will assume that these two subpopulations share their growth parameter, u, and process variance, σ^2, since they share a similar environment and prey base. However we postulate that because of fidelity to natal rookeries for breeding, animals do not move much year-to-year between the north and south and the two subpopulations are independent.

We need to write the state-space model to reflect this population structure. There are two subpopulations, x_n and x_s, and they have the same growth rate u:

$$\begin{bmatrix} x_{n,t} \\ x_{s,t} \end{bmatrix} = \begin{bmatrix} x_{n,t-1} \\ x_{s,t-1} \end{bmatrix} + \begin{bmatrix} u \\ u \end{bmatrix} + \begin{bmatrix} e_{n,t} \\ e_{s,t} \end{bmatrix} \tag{8.7}$$

We specify that they are independent by specifying that their year-to-year population fluctuations (their process error) come from a multivariate normal with no covariance:

$$\begin{bmatrix} e_{n,t} \\ e_{s,t} \end{bmatrix} \sim MVN\left(mean = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, varcov = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} \right) \tag{8.8}$$

For the observation process, we use a matrix to associate the regions with their respective x_n and x_s values:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n,t} \\ x_{s,t} \end{bmatrix} + \begin{bmatrix} \epsilon_{1,t} \\ \epsilon_{2,t} \\ \epsilon_{3,t} \\ \epsilon_{4,t} \\ \epsilon_{5,t} \end{bmatrix} \tag{8.9}$$

8.4.1 Specifying the `PopMARSS` arguments

We need to change the **Z** constraint to specify that there are 2 subpopulations (north and south), and that regions 1 and 2 are in the north subpopulation and regions 3,4 and 5 are in the south subpopulation:

```
Z.constraint = as.factor(c(1,1,2,2,2))
```

```
U.constraint = "equal"
Q.constraint = "diagonal and equal"
```

We want to specify that the u's are the same for each subpopulation and that **Q** is diagonal with equal σ^2's. To do this, we set

```
Z.constraint = as.factor(c(1,1,2,2,2))
```

```
U.constraint = "equal"
Q.constraint = "diagonal and equal"
```

This says that there is one u and one σ^2 parameter and both subpopulations share it (if we wanted the u's to be different, we would use `U.constraint="unequal"` or leave off the **U** constraint since the default behavior is `U.constraint="unequal"`).

Now we fit this model to the data and pass in the new constraints:

```
Z.constraint = as.factor(c(1,1,2,2,2))
U.constraint = "equal"
Q.constraint = "diagonal and equal"
R.constraint = "diagonal and equal"
kem3 = PopMARSS(dat, constraint=list(Z=Z.constraint,
   R=R.constraint, U=U.constraint, Q=Q.constraint))
```

```
Model Structure is
m: 2 state process(es)
n: 5 observation time series

Group 1 :
SJF EBays HC

Group 2 :
SJI PSnd

A :  scaling
B :  identity
Q :  diagonal and equal
R :  diagonal and equal
U :  equal
x0:  unconstrained
Converged in 25 interations. Max.iter was 5000.
```

The output tells us the structure of the model that was fit to the data and how long it took to fit the model. We estimated 2 initial x's, 1 process variance, 1 U, 3 A's, and 1 observation variance. So $K = 8$ parameters. The Kalman filter requires an initial condition ($t = 0$) for each x time series. When $m < n$, the number of A's estimated is $n - m$ since one of the A's for each state process will be set to 0. The AIC is `2*8 - 2*kem3$logLik`. Fig. 8.4 shows the residuals for the 2 subpopulations case. The residuals look better (more cloud-like) but the Hood Canal residuals are still temporally correlated.

Example 8.3 (Use `PopMARSS` to fit a model with North and South subpopulations)

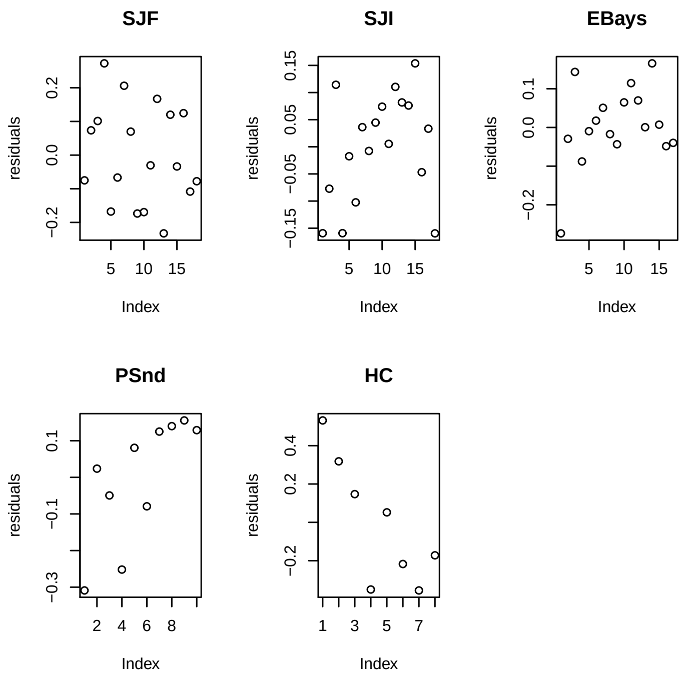

Fig. 8.4. The residuals for the analysis with a North and South subpopulation. The plots of the residuals should not have trends with time. Compare with the residuals for the analysis with one subpopulation.

Example 8.3 code

Type `show.doc(MARSS, Case_study_2.R)` to open a file in R with all the example code.

```
#fit model
Z.constraint = as.factor(c(1,1,2,2,2))
U.constraint = "equal"
Q.constraint = "diagonal and equal"
R.constraint = "diagonal and equal"
kem3 = PopMARSS(dat, constraint=list(Z=Z.constraint,
   R=R.constraint, U=U.constraint, Q=Q.constraint))
#plot residuals
plotdat = t(dat); plotdat[plotdat == -99] = NA;
matrix.of.biases = matrix(kem3$par$A,
  nrow=nrow(plotdat),ncol=ncol(plotdat),byrow=T)
par(mfrow=c(2,3))
for(i in 1:n){
        j=c(1,1,2,2,2)
        xs = kem3$states[j[i],]
        resids = plotdat[,i]-matrix.of.biases[,i]-xs
        plot(resids[!is.na(resids)],ylab="residuals")
        title(legendnames[i])
        }
par(mfrow=c(1,1))
```

8.5 Using PopMARSS to fit other population and observation error structures

Now work through a number of different structures and fill out the table at the back of this worksheet. At the end you'll see how your estimation of the mean population growth rate varies under different assumptions about the population and the data. All these analyses assume that the observation variances are unique at each site.

Example 8.4 (Five subpopulations)

Analyze the data using a model with five subpopulations, where each site is sampling one of the subpopulations. Assume that the subpopulation are independent (diagonal $\mathbf{Q}$*), however let each subpopulation share the same population parameters,* u *and* σ^2*. The Example 8.4 code shows how to set the* `PopMARSS` *arguments for this case. You can change* `R.constraint="diagonal and equal"` *to make all the observation variances equal.*

Example 8.4 code

Type `show.doc(MARSS, Case_study_2.R)` *to open a file in R with all the example code.*

```
Z.constraint=as.factor(c(1,2,3,4,5))
U.constraint="equal"
Q.constraint="diagonal and equal"
R.constraint="diagonal and unequal"
kem = PopMARSS(dat, constraint=list(Z=Z.constraint,
      U=U.constraint, Q=Q.constraint, R=R.constraint) )
```

Example 8.5 (Two subpopulations but different divisions)

Analyze the data using a model that assumes that the Strait of Juan de Fuca and San Juan Islands sites represent a Northern Puget Sound subpopulation, while the other three sites represent a Southern Puget Sound subpopulation. This time assume that each population trajectory (north and south) has different population parameters, u *and* σ^2 *and that each of the five sampling sites has a different observation variance. Try to write your own code for Examples 5-7. If you get stuck (or want to check your work, you can open* *a script file with all the Case Study 2 examples by typing* `show.doc(MARSS, Case_study_2.R)` *at the R command line.*

Example 8.6 (Hood Canal treated separately but covaries with others)

Analyze the data using a model with two subpopulations with the divisions being Hood Canal versus everywhere else. Set

```
Q.constraint = "equalvarcov"
```

to make all the subpopulations covary in time but with equal covariances and variances.

Example 8.7 (Three subpopulations with shared parameter values)

Analyze the data using a model with three subpopulations as follows: North (sites 1 and 2), South (sites 3 and 4), Hood Canal (site 5). You can specify that some subpopulations share parameters while others don't. You do this by using a vector of factors for the constraints:

```
Q.constraint = as.factor(c("coastal", "interior", "interior"))
U.constraint = as.factor(c("puget sound", "puget sound", "hood canal"))
R.constraint = as.factor(c("boat","boat","plane","plane","plane"))
```

When `Q.constraint` *and* `U.constraint` *are vectors (passed in as a factor), as above, they specify which* X*'s share parameter values. The factors must be a vector of length* m*, where* m *is the number of* X*'s. The* i*-th factor corresponds to the* i*-th* X*. In the example above, we specified that* X_1 *has its own process variance* Q *(which we named "coastal") and* X_2 *and* X_3 *share a process variance value (which we named "interior"). For the long-term trends, we specified that* X_1 *and* X_2 *share a long-term trend ("puget sound") while* X_3 *is allowed to have a separate trend ("hood canal").*

When `R.constraint` *is vector of factors, it specifies which* Y*'s have the same observation variance. We need a* 1×5 *vector here because we need to specify a value for each observation time series (there are 5). Here we imagine that observation time series 1 and 2 are boat surveys while the others are plane surveys and we want to allow the variances to differ based on methodology.*

8.6 Discussion

Case Study 2 shows you how to combine multiple datasets that are measuring the same underlying process and fit those data using a multivariate state-space framework. This allows you to combine data sets and use all the available data. You can also combine data that are discontinuous; that is data that don't overlap in time. For example, if you have data from one type of monitoring program in one set of years and then data from a different program starting in some later years, you can still easily estimate the population dynamics parameters using both sets of data.

There are a number of corners that we cut in order to have an example that runs quickly for a workshop:

- We ran the code starting from one initial condition. For a real analysis, you should start from a large number of random initial conditions and use the one that gives the highest likelihood. Since the EM algorithm is a "hill-climbing" algorithm, this ensures that it does not get stuck on a local maxima. `PopMARSS` will do this for you if you pass it the argument `Monte-CarloInitOptions=list(MCInit=TRUE)`. This will use a Monte Carlo routine to try many different initial conditions. See the help file on `PopMARSS` for more information (by typing `?PopMARSS` at the *R* prompt).
- We assume independent observation and process errors. Depending on your system, observation errors may is driven by large-scale environmental factors (temperature, tides, prey locations) that would cause your observation errors to covary across regions. If your observation errors strongly covary between regions and you treat them as independent, this could be bad for your analysis. The current EM code will not handle covariance in $\mathbf{R}$ when there are missing data, but even it did, separating covariance across observation versus process errors will require much data (to have any power). In practice, the first step is to think hard about what drives sightability for your species and what are the relative levels of process and observation variance. You may be able to change your data in a way that will make the observation errors independent–for example, using data from different months or defining your "regions"
- The `PopMARSS` argument `EMOptions` specifies the options for the EM algorithm. We left the default tolerance, `EMtol=0.01`. You'll want to set this lower, e.g. `EMtol=0.0001`, for a real analysis. You'll need to up the `max.EMiter` argument correspondingly.
- We used the large-sample computation for AIC instead of a bootstrap AIC that is designed to correct for small sample size in state-space models. The bootstrap metric, AICb, takes a long time to run (use the call `MARSSaic(kem, output=c("AICb"))` to compute AICb. We could have shown AICc, which is the small-sample size corrector for non-state-space models. Type `kem$AICc` to get that.

Finally, in a real (maximum-likelihood) analysis, one needs to be careful not to dredge the data. The temptation is to look at the data and pick a population structure that will fit that data. This can lead to including models in your analysis that have no biological basis. In practice, we spend a lot of time discussing the population structure with biologists working on the species and review all the biological data that might tell us what are reasonable structures. From that, a set of model structures to use are selected. Other times, a particular model structure needs to be used because the population structure is not in question rather it is a matter of using that pre-specified structure and using all the data to get parameter estimates for forecasting (U, Q, R). Finally, other times, one wants to have a measure of the support that the observed data give to all possible different population structures. That is a Bayesian question ($P(\Theta|data)$) and we would fit a model where $\mathbf{Q}$ is unconstrained and look at the posterior distribution of the elements in $\mathbf{Q}$.

Results table

Ex.		pop. growth rate `kem$par$U`	process variance `kem$par$Q`	K `kem$ params`	log-like `kem$num. logLik`	AIC `kem$AIC`
1	one population different obs. vars uncorrelated					
2	one population identical obs vars uncorrelated					
3	N+S subpops identical obs vars uncorrelated;					
4	5 subpops unique obs vars u's + σ^2's identical					
5	N+S subpops unique obs vars u's + σ^2's identical					
6	PS + HC subpops unique obs vars u's + σ^2's unique					
7	N + S + HC subpops unique obs vars u's + σ^2's unique					

For AIC, only the relative differences matter. A difference of 10 between two AIC means substantially more support for the model with lower AIC. A difference of 30 or 40 between two AICs is very large.

Questions

1. Do different assumptions about whether the measurement error variances are all identical versus different affect your estimate of the trend? You may want to rerun cases 3-7 with the `R.constraint` changed. `R.constraint="diagonal and unequal"` means measurement variances all different versus `"diagonal and equal"`.
2. Do assumptions about the underlying structure of the population affect your estimates of trend? Structure here means number of subpopulations and which areas are in which subpopulation. Try changing 'state parameters differ' to 'state params identical' for examples 5-7.

3. The CIs for the first two analyses are very tight because the estimate process variance was very small, `kem1$par$Q`. Why do you think σ^2 was forced to be so small? [Hint: We are forcing there to be 1 and only 1 true process and all the observation time series have to fit that one time series. Look at the AICs too.]

9

Case Study 3: Using MARSS models to identify spatial population structure and covariance

9.1 The problem

In this case study, we use time series of observations from 9 sites along the west coast to examine large-scale spatial structure in harbor seals (Jeffries et al., 2003). Harbor seals are distributed along the west coast of the US from California to Washington. The populations in Oregon and Washington have been surveyed for > 25 years at a number of haul-out sites (Figure 9.1). In general, these populations have been increasing steadily since the 1972 (Marine Mammal Protection Act). It remains unknown whether they are at carrying capacity.

For management purposes, 2 stocks are recognized; the coastal stock consists of 4 sites (Northern/Southern Oregon, Coastal Estuaries, Olympic Peninsula), and the inland WA stock consists of the remaining 5 sites (Figure 9.1). Subtle differences exist in the demographics across sites (e.g. pupping dates), however mtDNA analyses and tagging studies have suggested that these sites may be structured on a much larger scale. Harbor seals are known for strong site fidelity, but at the same time travel large distances to forage.

Our goal for this case study is to address the following questions about spatial structure: 1) Does population abundance data support the existing management boundaries, or are there alternative groupings that receive more support? and 2) Does the Hood Canal site represent a distinct subpopulation? Type `show.doc(MARSS, Case_study_3.R)` to open a file in R with all R code to get you started on the analyses in this chapter.

9.2 Analysis for question 1: how many distinct subpopulations?

For this analysis, we will analyze the support for five hypotheses about the population structure. These do not represent all possible structures but in-

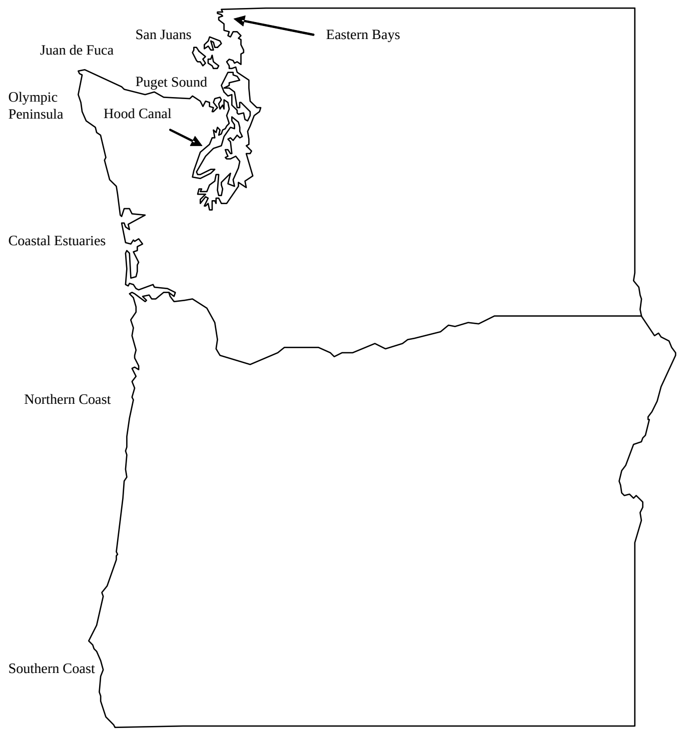

Fig. 9.1. Map of spatial distribution of 9 harbor seal sites in Washington and Oregon.

stead represent those that are considered most biologically plausible given the geography and the behavior of harbor seals.

Hypothesis 1 Sites are grouped by stock ($m = 2$), unique process variances
Hypothesis 2 Sites are grouped by stock ($m = 2$), same process variance
Hypothesis 3 Sites are grouped by state ($m = 2$), unique process variances
Hypothesis 4 Sites are grouped by state ($m = 2$), same process variance
Hypothesis 5 All sites are part of the same panmictic population ($m = 1$)

Aerial survey methodology has been relatively constant across time and space, and we will assume that all sites have the same constant (and independent) observation error variance for all sites.

9.2.1 Specify the design, Z, matrices

Write down the **Z** matrices for the hypotheses. Hint: Hypothesis 1 and 2 have the same **Z** matrix, Hypothesis 3 and 4 have the same **Z** matrix and Hypothesis 5 is a column of 1s.

$$\begin{array}{r|cc|cc|c|}
 & \multicolumn{2}{c}{\text{H 1 and 2}} & \multicolumn{2}{c}{\text{H 3 and 4}} & \multicolumn{1}{c}{\text{H 5}} \\
 & \multicolumn{2}{c}{\mathbf{Z}} & \multicolumn{2}{c}{\mathbf{Z}} & \multicolumn{1}{c}{\mathbf{Z}} \\
 & \text{subpop} & \text{subpop} & \text{subpop} & \text{subpop} & \text{subpop} \\
 & 1 & 2 & 1 & 2 & 1 \\
\texttt{Coastal Estuaries} & & & & & \\
\texttt{Olympic Peninsula} & & & & & \\
\texttt{Str. Juan de Fuca} & & & & & \\
\texttt{San Juan Islands} & & & & & \\
\texttt{Eastern Bays} & & & & & \\
\texttt{Puget Sound} & & & & & \\
\texttt{Hood Canal} & & & & & \\
\texttt{OR North Coast} & & & & & \\
\texttt{OR South Coast} & & & & & \\
\end{array}$$

Next you need to specify the constraints argument so that `PopMARSS` knows the structure of your **Z**'s. The Z constraint will be a vector of factors, i.e. it will have the form `as.factor(c(....))`.

- Hypothesis 1 and 2: `Z.constraint=`
- Hypothesis 3 and 4: `Z.constraint=`
- Hypothesis 5: `Z.constraint=`

9.2.2 Specify the grouping arguments

For this case study, we will assume that subpopulations share the same growth rate. What should `U.constraint` be for each hypothesis? To specify shared u parameters (for X_i), `U.constraint` is set as a length m vector of factors and specifies which subpopulations share their u parameter. Written in R it takes the form `as.factor(c(#,#,...))`

- Hypothesis 1-4: `U.constraint=`
- Hypothesis 5: `U.constraint=`

What about `Q.constraint`? To specify a diagonal **Q** matrix with shared values along the diagonal, `Q.constraint` is set as a length m vector of factors. The vector specifies which X_i's share their process variance parameter. Look at each hypothesis (above) and write down the corresponding `Q.constraint`.

- Hypothesis 1: `Q.constraint=`

- Hypothesis 2: `Q.constraint=`
- Hypothesis 3: `Q.constraint=`
- Hypothesis 4: `Q.constraint=`
- Hypothesis 5: `Q.constraint=`

Lastly, specify `R.constraint`. As we mentioned above, we will assume that the observation errors are independent and the observation variance is the same across sites. You can specify this constraint either as a text string or as a n length vector of factors.

- Hypothesis 1-5: `R.constraint=`

9.2.3 Fit models and summarize results

Fit each model for each hypothesis to the seal data (look at the script `Case_Study_3.r` for the code to load the data). Each call to `PopMARSS` will look like `kem = PopMARSS(sealData, constraint=list(Z = Z.constraint,`

`Q = Q.constraint, R = R.constraint, U = U.constraint))` Fill in the

following table, by fitting the five state-space models – that you have defined for the five hypotheses – to the harbor seal data (using `PopMARSS`). Use the `Case_Study_3.r` script so you don't have to type in all the commands.

H	pop. growth rate `kem$par$U`	process variance `kem$par$Q`	obs. variance `kem$par$R`	K `kem$num.params`	log-like. `kem$logLik`	AIC `kem$AIC`
1						
2						
3						
4						
5						

9.2.4 Interpret results for question 1

What do these results indicate about the process error grouping, and spatial grouping? A lower AIC means a more parsimonious model (highest likelihood given the number of parameters). A difference of 10 between AICs is large, and means the model with the higher AIC is unsupported relative to the model with lower AIC.

Extra analysis (if you have time): Do your results change if you assume that observation errors are independent but have unique variances? The 9 sites have different numbers of haul-outs and so the observation variances might be different. Repeat the analysis with unique observation variances for each site (this means changing `R.constraint`). You can also try the analysis with temporally co-varying subpopulations (good and bad years correlated) by setting `Q.constraint="unconstrained"` or `Q.constraint="equalvarcov"`.

9.3 Analysis for question 2: Is Hood Canal separate?

The Hood Canal site may represent a distinct population, and has recently been subjected to a number of catastrophic events (hypoxic events, possibly leading to reduced prey availability, and several killer whale predation events, removing up to 50% of animals per occurrence). Build four models, assuming that each site (other than Hood Canal) is assigned to its current management stock, but Hood Canal is allowed to be a different subpopulation ($m = 3$). Again, assume observation error is independent and constant across sites.

Hypothesis 1 Subpopulations have the same process variance and growth rate
Hypothesis 2 Each subpopulation has a unique process variance and growth rate
Hypothesis 3 Hood Canal has the same process variance, but different growth rate
Hypothesis 4 Hood Canal has unique process variance and unique growth rate

9.3.1 Specify the Z matrix and `Z.constraint`

The **Z** matrix for each hypothesis is the same. The coastal subpopulation consists of 4 sites (Northern/Southern Oregon, Coastal Estuaries, Olympic Peninsula), the Hood Canal subpopulation is the Hood Canal site, and the inland WA subpopulation consists of the remaining 4 sites. Thus $m = 3$ and **Z** is a 9×3 matrix:

$$\begin{array}{rc}
 & \begin{array}{ccc}\text{subpop} & \text{subpop} & \text{subpop}\\ 1 & 2 & 3\end{array} \\
\begin{array}{r}\texttt{Coastal Estuaries}\\ \texttt{Olympic Peninsula}\\ \texttt{Str. Juan de Fuca}\\ \texttt{San Juan Islands}\\ \texttt{Eastern Bays}\\ \texttt{Puget Sound}\\ \texttt{Hood Canal}\\ \texttt{OR North Coast}\\ \texttt{OR South Coast}\end{array} & \left[\begin{array}{ccc} & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{array}\right]
\end{array}$$

Then write down `Z.constraint` for this **Z**.

9.3.2 Specify which parameters are shared across which subpopulations

`U.groups` specifies which u are shared across subpopulations. Look at the hypothesis descriptions above which will specify whether subpopulations share their population growth rate or have unique population growth rates.

- Hypothesis 1: `U.constraint=`
- Hypothesis 2: `U.constraint=`
- Hypothesis 3: `U.constraint=`
- Hypothesis 4: `U.constraint=`

`U.constraint` will be a length m vector of factors. Once you have more than 2 subpopulations, it can get hard to keep straight which `U.constraint=` number goes to which subpopulation. It is best to sketch your **Z** matrix (which tells you which site in the rows corresponds to which subpopulation in the columns). Then remember that the elements of `U.constraint` correspond 1 to 1 with the columns of **Z**:

```
U.constraint=as.factor(c(col 1 Z, col 2 Z, col 3 Z, ..)).
```

Specify `Q.groups` showing which subpopulations share their process variance parameter.

- Hypothesis 1: `Q.constraint=`
- Hypothesis 2: `Q.constraint=`
- Hypothesis 3: `Q.constraint=`
- Hypothesis 4: `Q.constraint=`

`Q.constraint` will be a length m vector of factors. `R.constraint` is the same as for Question 1; the observation variances are the same for each site.

9.3.3 Fit the models and summarize results

Fit each model for each hypothesis to the seal data (look at the script `Case_Study_3.r` for the code to load the data). Each call to `PopMARSS` will look like

```
   kem = PopMARSS(sealData, constraint=list(Z = Z.constraint, Q =
Q.constraint, R = R.constraint, U = U.constraint))
```

H	pop. growth rate `kem$par$U`	proc. variance `kem$par$Q`	obs. variance `kem$par$R`	K `kem$num.params`	log-like `kem$logLik`	AIC `kem$AIC`
1						
2						
3						
4						

9.3.4 Interpret results for question 2

How do the residuals for the Hood Canal site compare from these models relative to the best model from Question 1? Hint: If you have the vector of estimated population states (`Xpred = t(kem$states)`) and the data (`Xobs = sealData`), the residuals for site i can be plotted in R as:

```
Xpred = t(kem$states)
Xobs = sealData
plot(Xpred[, Z.constraint[i]] - Xobs[,i],
      ylab="Predicted-Observed Data")
```

In R, if you have a matrix `Y[1:numYrs, 1:n]`, you can extract column j by writing `Yj = Y[,j]`.

Relative to the previous models from Question 1, do these scenarios have better or worse AIC scores (smaller AIC is better)? If you were to provide advice to managers, would you recommend that the Hood Canal population is a source or sink? What implications does this have for population persistence?

Code for Case Study 3
Type `show.doc(MARSS, Case_study_3.R)` to open a file in R with all the example code.

10

Case Study 5: Using state-space models to analyze noisy animal tracking data

10.1 A simple random walk model of animal movement

A simple random walk model of movement with drift but no correlation is

$$\begin{aligned} x_{1,t} &= x_{1,t-1} + u_1 + e_{1,t}, \quad e_{1,t} \sim Normal(0, \sigma_1^2) \\ x_{2,t} &= x_{2,t-1} + u_2 + e_{2,t}, \quad e_{2,t} \sim Normal(0, \sigma_2^2) \end{aligned} \tag{10.1}$$

where $x_{1,t}$ is the location at time t along one axis (in our case study, longitude) and $x_{2,t}$ is for another, generally orthogonal, axis (in our case study, latitude). We add errors to our observations of location:

$$\begin{aligned} y_{1,t} &= x_{1,t} + a_1 + \epsilon_{1,t}, \quad \epsilon_{1,t} \sim Normal(0, \eta_1^2) \\ y_{2,t} &= x_{2,t} + a_2 + \epsilon_{2,t}, \quad \epsilon_{2,t} \sim Normal(0, \eta_2^2), \end{aligned} \tag{10.2}$$

Together Equations 10.2 and 10.3 describe a MARSS model (now written in matrix form):

$$\mathbf{X}_t = \mathbf{X}_{t-1} + \mathbf{U} + \mathbf{E}_t, \quad \mathbf{E}_t \sim MVN(0, \mathbf{Q}) \tag{10.3}$$

$$\mathbf{Y}_t = \mathbf{X}_t + \mathbf{A} + \boldsymbol{\eta}_t, \quad \boldsymbol{\eta}_t \sim MVN(0, \mathbf{R}). \tag{10.4}$$

10.2 The problem

Loggerhead sea turtles (*Caretta caretta*) are listed as threatened under the United States Endangered Species Act of 1973. Over the last ten years, a number of state and local agencies have been deploying ARGOS tags on loggerhead turtles on the east coast of the United States. We have data on eight individuals over that period. In this case study, we use some turtle data from the WhaleNet Archive of STOP Data, however we have corrupted this data severely by adding random errors in order to create a "Bad Tag" problem. We corrupted latitude and longitude data by errors (Figure 10.1) and it would appear that our sea turtles are becoming land turtles (at least part of the time).

For this case study, we will use PopMARSS to estimate true positions and speeds from the corrupted data. We will use a mapping package to plot the results: the maps package. If you have not already, install this package by selecting the 'Packages' menu and then 'Install packages' and then select maps. If you are on a Mac, remember to select "binaries" for the package type. Type show.doc(MARSS, Case_study_5.R) to open a file in R with all R code to get you started on the analyses in this chapter.

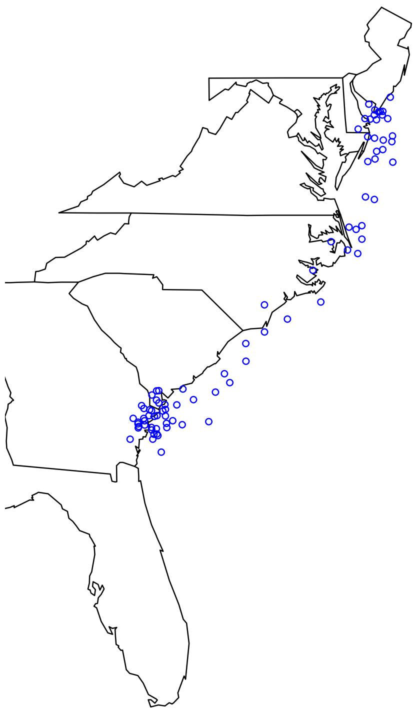

Fig. 10.1. Plot of the tag data from the turtle Big Mama. Errors in the location data make it seem that Big Mama has been moving overland.

10.3 Using the Kalman-EM algorithm to estimate locations from bad tag data

10.3.1 Read in the data and load `maps` package

Our noisy data are in `loggerheadNoisy`. They consist of daily readings of location (longitude/latitude). The data are recorded daily and `PopMARSS` requires an data entry for each day. If data are missing for a day, then the entries for lat and lon for that day should be -99. However, to make this case study run quickly, we have interpolated all missing values in the original, uncorrupted, dataset (`loggerhead`). The corrupted data look like so

```
loggerheadNoisy[1:6,]
```

```
   turtle month day year       lon      lat
1 BigMama     5  28 2001 -81.45989 31.70337
2 BigMama     5  29 2001 -80.88292 32.18865
3 BigMama     5  30 2001 -81.27393 31.67568
4 BigMama     5  31 2001 -81.59317 31.83092
5 BigMama     6   1 2001 -81.35969 32.12685
6 BigMama     6   2 2001 -81.15644 31.89568
```

and the file has data for 8 turtles:

```
levels(loggerheadNoisy$turtle)
```

```
[1] "BigMama"  "Bruiser"  "Humpty"   "Isabelle" "Johanna"
[6] "MaryLee"  "TBA"      "Yoto"
```

We will first analyze the position data for “Big Mama”. We put the data for “Big Mama” into variable `dat`. `dat` is transposed because we need time across the columns.

```
turtlename="BigMama"
dat = loggerheadNoisy[which(loggerheadNoisy$turtle==turtlename),5:6]
dat = t(dat) #transpose
```

10.3.2 Use `PopMARSS` to estimate the position of Big Mama

We will begin by specifying the structure of the MARSS model and then use `PopMARSS` to fit that model to the data for each individual. There are two state processes (one for latitude and the other for longitude). There is one observation time series for each so

```
Z.constraint=as.factor(c(1,2))
```

We’ll assume that the errors are independent and that there are different drift rates ($\mathbf{U}$), process variances ($\mathbf{Q}$) and measurement variances for latitude and longitude ($\mathbf{R}$). You can try model constraints if you wish.

```
U.constraint="unequal"
Q.constraint="diagonal and unequal"
R.constraint="diagonal and unequal"
```

Fit the model to the data:

```
kem = PopMARSS(dat, constraint=list(Z = Z.constraint,
      Q = Q.constraint, R = R.constraint, U = U.constraint))
```

10.3.3 Compare state estimates to the real positions

The real locations (from which `loggerheadNoisy` was produced by adding noise) are in `loggerhead`. In Figure 10.2, we compare the tracks estimated from the noisy data with the original, good, data (see the *R* script, `Case_Study_5.r` for the code to make this plot. There are only a few data points for the real data because the real tag data has many missing days.

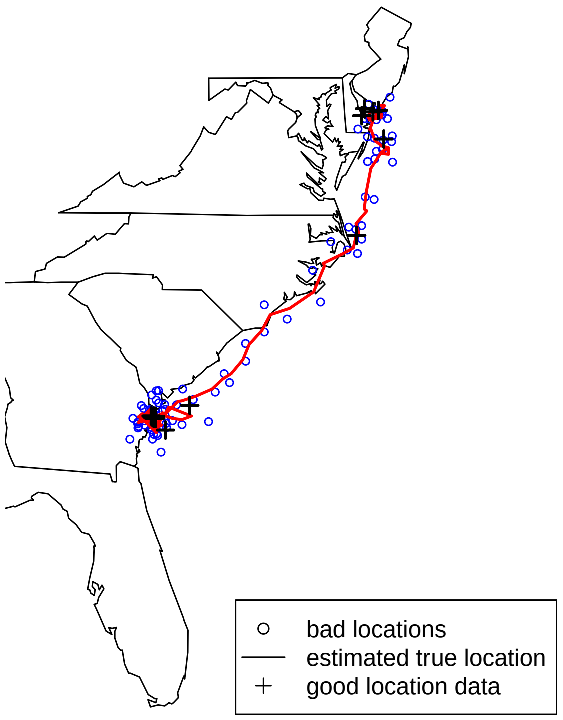

Fig. 10.2. Plot of the estimated track of the turtle Big Mama versus the good location data (before we corrupted it with noise).

10.3.4 Estimate speeds for each turtle

Turtle biologists designated one of these loggerheads "Big Mama," presumably for her size and speed. For each of the eight turtles, estimate the average miles traveled per day. To calculate the distance traveled by a turtle each day, you use the estimate (from `PopMARSS`) of the lat/lon location of turtle at day t and at day $t-1$. To calculate distance traveled in miles from lat/lon start and finish locations, we will use the function `GCDF` defined at the beginning of the *R* script, `Case_Study_5.r`):

```
distance[i-1]=GCDF(pred.lon[i-1],pred.lon[i],
                   pred.lat[i-1],pred.lat[i])
```

`pred.lon` and `pred.lat` are the predicted longitudes and latitudes from `PopMARSS`. To calculate the distances for all days, we put this through a `for` loop:

```
distance = array(-99, dim=c(dim(dat)[2]-1,1))
for(i in 2:dim(dat)[2])
   distance[i-1]=GCDF(pred.lon[i-1],pred.lon[i],pred.lat[i-1],pred.lat[i])
```

The command (`mean(distance)` gives us the average distance per day. We can also make a histogram of the distances traveled per day (Figure 10.3). Repeat the analysis done for "Big Mama" for each of the other turtles and fill out the speed table (Table 10.3.4). If you were given the opportunity to race these turtles, would you bet on Big Mama being the fastest?

Turtle	Estimated Speed
Big Mama	
Bruiser	
Humpty	
Isabelle	
Johanna	
Mary Lee	
TBA	
Yoto	

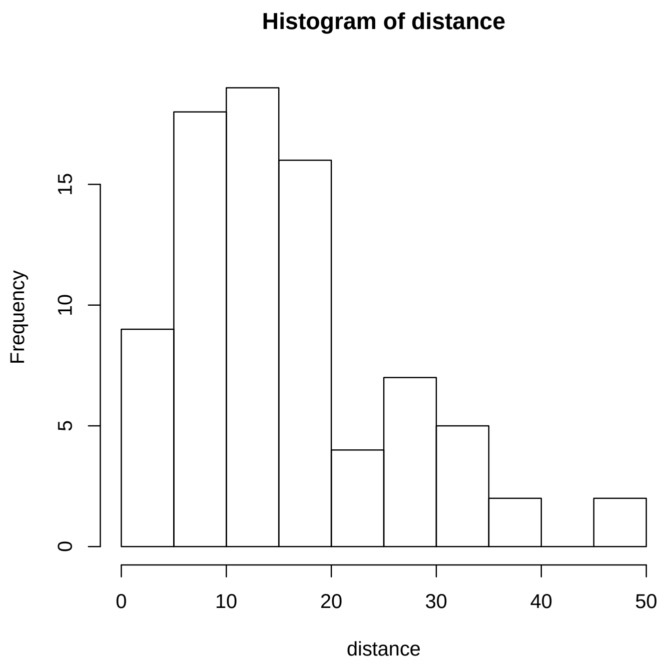

Fig. 10.3. Histogram of the miles traveled per day for Big Mama. Compare this to the estimate of miles traveled per day if you had not accounted for measurement errors. See the script file, `Case_Study_5.r`, for the code to this.

10.4 Comparing turtle tracks to proposed fishing areas

One of the greatest threats to the long term viability of loggerhead turtles is incidental take by net/pot fisheries. Add two proposed fishing areas to your turtle plots:

```
# the proposed fishery areas
lines(c(-77,-78,-78,-77,-77),
      c(33.5,33.5,32.5,32.5,33.5),col="red",lwd=2)
lines(c(-75,-76,-76,-75,-75),
      c(38,38,37,37,38),col="red",lwd=2)
```

Given that only one area can be chosen as a future fishery, what do your predicted movement trajectories for our eight turtles tell you?

10.5 Using `fields` to get density plots of locations

If you are comfortable programming in *R*, load the `fields` package. Make 3D density plots of predicted sea turtle locations. Which two areas appear to be most visited?

Include the confidence interval estimates for each location in this analysis. For this part of the exercise, we will assume that the confidence intervals are roughly the same as the probability intervals (Bayesian). We can assume that the error in latitude is independent from error in longitude. The `fields` package includes a couple different functions. One that might be useful here is `Tps()`, like in the example (`?fields`). To call `fields`, we need predictor variables (X), which can be random lon/lat pairs randomly drawn within the range of the data. The other requirement for `Tps()` is the response, y. If we think of each predicted state being a bivariate normal density, the response for each of our random pairs can be the density across all of the predicted states. There is code to help you get started in the *R* file, `Case_Study_5.r`.

10.6 Using specialized packages to analyze tag data

If you have real tag data to analyze, you should use a state-space modeling package that is customized for fitting MARSS models to that kind of data. The MARSS package does not have all the bells and whistles that you would want for analyzing tracking data, particularly tracking data in the marine environment. These are a couple *R* packages that we have come across for this purpose:

UKFSST http://www.soest.hawaii.edu/tag-data/tracking/ukfsst/
KFTRACK http://www.soest.hawaii.edu/tag-data/tracking/kftrack/

`kftrack` is a full-featured toolbox for analyzing tag data with extended Kalman filtering. It incorporates a number of extensions that are important for analyzing track data: barriers to movement such as coastlines and non-Gaussian movement distributions. With `kftrack`, you can use the real tag data which has big gaps, i.e. days with no location. `PopMARSS` will struggle with these data because it will estimate states for all the unseen days; `kftrack` only fits to the seen days.

To use `kftrack` to fit the turtle data, type

```
library(kftrack) # must be installed from a local zip file
loggerhead = loggerhead
# Run kftrack with the first turtle (BigMama)
turtlename = "BigMama"
model = kftrack(loggerhead[ which(loggerhead$turtle == turtlename), 2:6],
        fix.first=F, fix.last=F, var.struct="uniform")
```

To look at what the `kftrack` model consists of, type

```
 model
```

Code for Case Study 5
Type `show.doc(MARSS, Case_study_5.R)` to open a file in R with the example code.

A

Package MARSS: Object structures

A.1 Model objects: class ‘marssm’

Objects of class ‘marssm’ specify Multivariate Autoregressive State Space (MARSS) models. The `model` component of an ML estimation object (class ‘marssMLE’; see below) is an ‘marssm’ object. These objects have the following components:

data An optional matrix (not dataframe), in which each row is a time series (time across columns).
fixed A list with 8 matrices Z, A, R, B, U, Q, x0, V0, specifying which elements of each parameter are fixed.
free A list with 8 matrices Z, A, R, B, U, Q, x0, V0, specifying which elements of each parameter are to be estimated.
M An array of dim n x n x t (an n x n missing values matrix for each time point). Each matrix is diagonal with 0 at the i,i value if the i-th value of y is missing, and 1 otherwise.
miss.value Specifies missing value representation in the data.

The matrices in `fixed` and `free` work as pairs to specify the fixed and free elements for each parameter. See section 3. The dimensions for `fixed` and `free` matrices are as follows, where n is the number of sites and m is the number of subpopulations (state processes):

Z	n x m
B	m x m
U	m x 1
Q	m x m
A	n x 1
R	n x n
x0	m x 1
V0	m x m

Use `is.marssm()` to check whether an ‘marssm’ object is correctly specified. The`MARSS` package includes an `as.marssm()` method for wrapper objects of class ‘popWrap’ (see next section). We recommend that creators of new wrapper classes write new `as.marssm()` methods for their classes.

A.2 Wrapper objects: class ‘popWrap’

Wrapper objects of class ‘popWrap’ contain specifications and options for estimation of a MARSS model. A ‘popWrap’ object has the following components:

data A matrix (not dataframe), sites (rows) x time (columns), of observed population abundances. If the algorithm is to be applied to log-abundance, the log transformation should be done before the data is passed in.
m Number of subpopulations (state processes).
constraint Either a list with 8 string elements Z, A, R, B, U, Q, x0, V0 (see below for details), or string ‘ignore’.
fixed If `constraint = 'ignore'`, a list with 8 matrices Z, A, R, B, U, Q, x0, V0.
free If `constraint = 'ignore'`, a list with 8 matrices Z, A, R, B, U, Q, x0, V0.
inits A list with 7 matrices A, R, B, U, Q, x0, V0, specifying initial values for parameters. Dimensions are given in the class ‘marssm’ section.
miss.value Specifies missing value representation.
EMOptions List of estimation options for the EM algorithm, containing elements `max.EMiter, EMtol, iter.V0` and `debugEM`. See class ‘marssMLE’ section for details.
MonteCarloInitOptions List of options for Monte Carlo initialization, containing elements `MCInit, numInits, numInitSteps` and `bounds`. See class ‘marssMLE’ section for details.

Component `constraint` is a convenient way to specify model structure for certain common cases. If `constraint = 'ignore'`, both `fixed` and `free` must be provided. See the class ‘marssm’ section for how to specify fixed and free matrices. The wrapper function `PopMARSS()` calls `popWrap()` to create a ‘popWrap’ object, then `is.marssm()` to coerce this object to class ‘marssm’ for the estimation function.

The `popWrap()` function calls `checkPopWrap()` to check user inputs. Valid constraints are as follows; see section 4 for details. (The V0 constraint depends on the x0 constraint.)

A Must be string ‘scaling’.
B String ‘fixed’, ‘identity’, ‘unconstrained’, ‘diagonal and unequal’, or ‘diagonal and equal’. May also be vector of class factor specifying shared diagonal values.

Q String 'fixed', 'unconstrained', 'diagonal and unequal', 'diagonal and equal', or 'equalvarcov'. May also be vector of class factor specifying shared diagonal values.
R String 'fixed', 'unconstrained', 'diagonal and unequal', 'diagonal and equal', or 'equalvarcov'. May also be numeric vector of class factor specifying diagonal valuess.
U String 'fixed', 'unconstrained'='unequal', or 'equal'. May also be vector of class factor specifying shared growth rates.
x0 String 'fixed', 'unconstrained'='unequal', or 'equal'. May also be vector of class factor specifying shared initial conditions.
Z String 'fixed' or a vector of class factor specifying which **Y** time series correspond to which state time series (the **X**s).

A.3 ML estimation objects: class 'marssMLE'

Objects of class 'marssMLE' specify maximum likelihood estimation of a MARSS model. A minimal 'marssMLE' object contains components `model, start` and `control`, which must be present for estimation by functions like `MARSSkem()`.

model MARSS model specification (an object of class 'marssm').
start List with 7 matrices A, R, B, U, Q, x0, V0, specifying initial values for parameters. Dimensions are given in the class 'marssm' section.
control A list specifying estimation options.
max.EMiter Maximum number of EM iterations.
EMtol Optional tolerance for log-likelihood change. If log-likelihood decreases less than this amount relative to the previous iteration, the EM algorithm exits.
iter.V0 Maximum number of iterations for final likelihood calculation with V0 = 0.
debugEM If TRUE, a record will be created of each variable over all EM iterations.
MCInit Use Monte Carlo initialization?
numInits Number of random initial value draws.
numInitSteps Number of iterations for each initial value draw.
bounds Bounds on the uniform distributions from which initial values will be drawn. (Note that bounds for the covariance matrices Q and R, which require positive values, are specified in logs.)
silent Suppresses printing of progress bar and convergence information.

`MARSSkem()` appends the following components to the 'marssMLE' object:

method A string specifying the estimation method ('kem' for estimation by `MARSSkem()`).
par A list with 8 matrices of estimated parameter values Z, A, R, B, U, Q, x0, V0.

kf A list containing Kalman filter/smoother output. See section 5.1.
numIter Number of iterations required for convergence.
convergence Convergence status.
logLik Log-likelihood.

Several functions append additional components to the 'marssMLE' object passed in. These include:

`MARSSaic` Appends `AIC, AICc, AICbb, AICbp` and/or `AICi`, depending on the AIC flavors requested.
`MARSShessian` Appends `Hessian, gradient, parMean` and `parSigma`.
`MARSSparamCIs` Appends `par.se, par.upCI` and `par.lowCI`.

B

Package MARSS: Base functions and wrappers

Package MARSS includes functions for estimating Multivariate Autoregressive State Space models, obtaining confidence intervals for parameters, and calculating Akaike's Information Criterion (AIC) for model selection. To make the package both flexible and easy to use, it is designed in two levels. At the base level, the programmer can interact directly with the estimation functions, using two kinds of R objects: objects of the model specification class 'marssm', and objects of estimation classes such as 'marssMLE'. At the user level, wrapper functions allow model estimation with just one function call, hiding the details for ease of use. Users create models in an intuitive way by specifying constraints; the wrapper functions then convert these constraints into the object structures required by the estimation functions, performing error checking as necessary.

The two-level package structure allows new users convenient access to the underlying functions, while maintaining flexibility to incorporate different applications and algorithms. Developers can use the base object types to write new wrappers for their own modeling applications.

The MARSS package provides the wrapper function `PopMARSS()`, which was developed specifically for use in ecological applications including:

- Modeling the structure of population dynamics of a single species measured at multiple sites.
- Modeling community interactions of several different species.

To use the wrapper, the user specifies a model by supplying the constraint argument to `PopMARSS()`, using the method argument to specify an estimation method. Optionally, the user may provide initial values for the free parameters, and specify estimation options; for details see the `PopMARSS()` help file. The function returns an object containing the model, parameter values and estimation details. The user may pass the returned object to `MARSSboot()`, which generates bootstrap parameter estimates, or to `MARSSaic()`, which calculates various versions of AIC for model selection.

Figure 1 shows the underlying base level operations `PopMARSS()`performs. The function creates a wrapper object of class 'popWrap'. It then calls the `as.marssm( )` method for 'popWrap' to create a `marssm` model specification object from the constraints provided. This model object, initial values and control information are the minimal information required by the estimation functions, and are combined into an object of class appropriate for the estimation method. The estimation function adds to this object the estimated parameter values, estimation details, and other function-specific components, and then returns the augmented object.

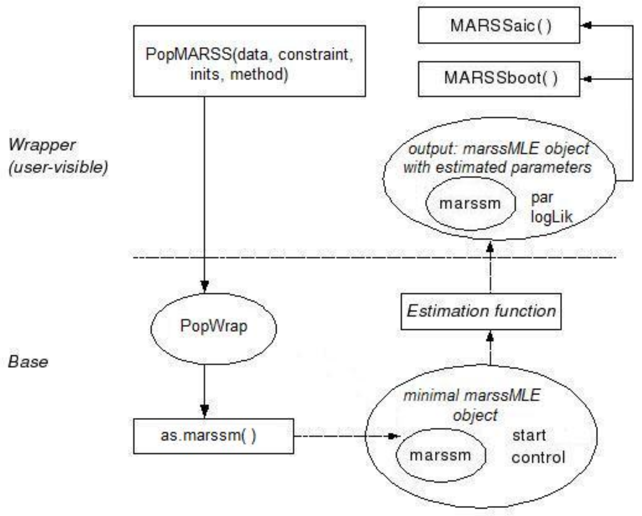

Fig. B.1. Two-level structure of the MARSS package. Rectangles represent functions; ovals represent objects.

References

Biernacki, C., Celeux, G., and Govaert, G. (2003). Choosing starting values for the em algorithm for getting the highest likelihood in multivariate gaussian mixture models. *Computational Statistics and Data Analysis*, 41(3-4):561–575.

Brockwell, P. J. and Davis, R. A. (1991). *Time series: theory and methods*. Springer-Verlag, New York, NY.

Cavanaugh, J. and Shumway, R. (1997). A bootstrap variant of aic for state-space model selection. *Statistica Sinica*, 7:473–496.

Dempster, A., Laird, N., and Rubin, D. (1977). Likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Dennis, B., Munholland, P. L., and Scott, J. M. (1991). Estimation of growth and extinction parameters for endangered species. *Ecological Monographs*, 61:115–143.

Dennis, B., Ponciano, J. M., Lele, S. R., Taper, M. L., and Staples, D. F. (2006). Estimating density dependence, process noise, and observation error. *Ecological Monographs*, 76(3):323–341.

Ellner, S. P. and Holmes, E. E. (2008). Resolving the debate on when extinction risk is predictable. *Ecology Letters*, 11:E1–E5.

Gerber, L. R., Master, D. P. D., and Kareiva, P. M. (1999). Grey whales and the value of monitoring data in implementing the u.s. endangered species act. *Conservation Biology*, 13:1215–1219.

Ghahramani, Z. and Hinton, G. E. (1996). Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science.

Harvey, A. C. (1989). *Forecasting, structural time series models and the Kalman filter*. Cambridge University Press, Cambridge, UK.

Harvey, A. C. and Shephard, N. (1993). Structural time series models. In Maddala, G. S., Rao, C. R., and Vinod, H. D., editors, *Handbook of Statistics, Volume 11*. Elsevier Science Publishers B V, Amsterdam.

Hinrichsen, R. (2009). Population viability analysis for several populations using multivariate state-space models. *Ecological Modelling*, 220(9-10):1197–1202.

Hinrichsen, R. and Holmes, E. E. (2009). Using multivariate state-space models to study spatial structure and dynamics. In Cantrell, R. S., Cosner, C., and Ruan, S., editors, *Spatial Ecology*. CRC/Chapman Hall.

Holmes, E. E. (2001). Estimating risks in declining populations with poor data. *Proceedings of the National Academy of Sciences of the United States of America*, 98(9):5072–5077.

Holmes, E. E. (2010). Derivation of the em algorithm for constrained and unconstrained marss models.

Holmes, E. E., Sabo, J. L., Viscido, S. V., and Fagan, W. F. (2007). A statistical approach to quasi-extinction forecasting. *Ecology Letters*, 10(12):1182–1198.

Jeffries, S., Huber, H., Calambokidis, J., and Laake, J. (2003). Trends and status of harbor seals in washington state 1978-1999. *Journal of Wildlife Management*, 67(1):208–219.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45.

Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimation of linear dynamical systems. *Journal of AIAA*, 3:1445–1450.

Schweppe, F. C. (1965). Evaluation of likelihood functions for gaussian signals. *IEEE Transactions on Information Theory*, IT-r:294–305.

Shumway, R. and Stoffer, D. (2006). *Time series analysis and its applications*. Springer-Science+Business Media, LLC, New York, New York, 2nd edition.

Shumway, R. H. and Stoffer, D. S. (1982). An approach to time series smoothing and forecasting using the em algorithm. *Journal of Time Series Analysis*, 3(4):253–264.

Staples, D. F., Taper, M. L., and Dennis, B. (2004). Estimating population trend and process variation for pva in the presence of sampling error. *Ecology*, 85(4):923–929.

Stoffer, D. S. and Wall, K. D. (1991). Bootstrapping state-space models: Gaussian maximum likelihood estimation and the kalman filter. *Journal of the American Statistical Association*, 86(416):1024–1033.

Taper, M. L. and Dennis, B. (1994). Density dependence in time series observations of natural populations: estimation and testing. *Ecological Monographs*, 64(2):205–224.

Ward, E. J., Chirakkal, H., GonzÃąlez-SuÃąrez, M., Aurioles-Gamboa, D., Holmes, E. E., and Gerber, L. . (2009). Inferring spatial structure from time-series data: using multivariate state-space models to detect metapopulation structure of california sea lions in the gulf of california, mexico. *Journal of Applied Ecology*, 1(47):47–56.

Index